Extracting Unsatisfiable Cores for LTL via Temporal Resolution

(full version; r828, November 15, 2013)

Viktor Schuppan
Email: Viktor.Schuppan@gmx.de

***Abstract*—Unsatisfiable cores (UCs) are a well established means for debugging in a declarative setting. Still, tools that perform automated extraction of UCs for LTL are scarce. Using resolution graphs to extract UCs is common in many domains. In this paper we construct and optimize resolution graphs for temporal resolution as implemented in the temporal resolution-based solver `TRP++` and we use them to extract UCs for propositional LTL. We implement our method in `TRP++`, and we experimentally evaluate it. Source code of our tool is available.**

***Index Terms*—LTL; unsatisfiable cores; vacuity; temporal resolution;**

I. Introduction

A. Motivation

Debugging is an activity that many hardware and software developers spend a fair amount of time on. When faced with some input that induces an undesired behavior it is typically suggested to minimize that failure-inducing input in order to simplify identification of the problem (e.g., [ZH02]). Corresponding research has been performed, e.g., in linear programming (e.g., [CD91]), constraint satisfaction (e.g., [Bak+93]), compilers (e.g., [Wha94]), SAT (e.g., [BS01]), declarative specifications (e.g., [Shl+03]), and LTL satisfiability (e.g., [Sch12]) and realizability (e.g., [Cim+08]).

LTL [Pnu77,Eme90] and its relatives are important specification languages for reactive systems (e.g., [EF06]) and for business processes (e.g., [PA06]). Experience in verification as well as in synthesis has lead to specifications themselves becoming objects of analysis. Beer et al. report [Bee+01] that in their experience "[...] during the first formal verification runs of a new hardware design, typically 20 % of formulas are found to be trivially valid, and that trivial validity always points to a real problem in either the design or its specification or environment.". In a work on LTL synthesis [Blo+07] Bloem et al. state that "[...] writing a complete formal specification [...] was not trivial." and "Although this approach removes the need for verification [...] the specification itself still needs to be validated.".

Typically, a specification is expected to be satisfiable. If it turns out to be unsatisfiable, finding a reason for unsatisfiability can help with the ensuing debugging. Frequently, such reason for unsatisfiability is taken to be a part of the unsatisfiable specification that is by itself unsatisfiable (e.g., [Sch12,Bak+93,CD91]); this is called an unsatisfiable core (UC) (e.g., [Sch12,GN03,ZM03b,Hoo99]). Less simplistic ways to examine an LTL specification ϕ exist [Pil+06], and understanding their results also benefits from availability of UCs. First, one can ask whether a certain scenario ϕ', given as an LTL formula, is permitted by ϕ. That is the case iff $\phi \wedge \phi'$ is satisfiable. Second, one can check whether ϕ ensures a certain LTL property ϕ''. ϕ'' holds in ϕ iff $\phi \wedge \neg\phi''$ is unsatisfiable. In the first case, if the scenario turns out not to be permitted by the specification, a UC can help to understand which parts of the specification and the scenario are responsible for that. In the second case a UC can show which parts of the specification imply the property. Moreover, if there are parts of the property that are not part of the UC, then those parts of the property could be strengthened without invalidating the property in the specification; i.e., the property is vacuously satisfied (e.g., [Sim+10,Bee+01,KV03,Arm+03, GC04,Fis+08,Kup06]). UCs are therefore an important part of design methods for embedded systems (e.g., [Pil+06]) as well as for business processes (e.g., [Awa+12]). Note that specifications of real world systems may be 100s of pages long (e.g., [Chi+10]). Hence, providing automated support for obtaining a UC in case such a specification turns out to be unsatisfiable is crucial.

UCs also have applications in avoiding the exploration of parts of a search space that can be known not to contain a solution for reasons "equivalent" to the reasons for previous failures (e.g., [Cla+03,Cim+07]) and in certifying the correctness of a result of unsatisfiability (e.g., [VG02,GN03, ZM03b]). While our results also benefit these applications, we focus on debugging below.

Despite their relevance interest in UCs for LTL has been somewhat limited (e.g., [Cim+07,Sch12,HH11,HSH12]). In particular, publicly available tools that automatically extract fine-grained UCs for propositional LTL are scarce.

B. Temporal Resolution as a Basis

Extracting UCs is often possible using any solver for the logic under consideration by weakening subformulas one by one and using the solver to test whether the weakened formula is still unsatisfiable (e.g., [MS10]). While that is simple to implement, repeated testing for preservation of unsatisfiability may impose a significant run time burden. Hence, it is interesting to investigate methods to extract UCs from a single run of a solver. Extracting UCs from resolution graphs is common in SAT (e.g., [VG02]). A resolution method (e.g., [BG01, Rob65]) for LTL, temporal resolution (TR), was suggested

by Fisher [Fis91,FDP01] and implemented in TRP++ [HK04, HK03,trp++]. TRP++ is available as source code [trp++].

TR lends itself as a basis for extracting UCs for LTL for two reasons. First, the TR-based solver TRP++ proved to be competitive in a recent evaluation of solvers for LTL satisfiability, in particular on unsatisfiable instances (see pp. 51–55 of the full version of [SD11]). Second, a TR proof naturally induces a resolution graph, which provides a clean framework for extracting a UC. Note, that while the BDD-based solver NuSMV [Cim+02] also performed well on unsatisfiable instances in [SD11], the BDD layer makes extraction of a UC more involved. On the other hand, the tableau-based solvers LWB [Heu+95] and pltl [pltl] provide access to a proof of unsatisfiability comparable to TR, yet tended to perform worse on unsatisfiable instances in [SD11].

C. Contributions

In this paper we make the following contributions. We construct resolution graphs for TR for propositional LTL as implemented in TRP++, and we use them to extract UCs. Note that TR is significantly more complex than propositional resolution. Hence, we use the specifics of TR in TRP++ to optimize the construction of resolution graphs. The temporal aspect also allows to extract more fine-grained information from the resolution graph; this is exploited in a companion paper [Sch13], which this paper provides the basis for. We implement our method in TRP++, and we experimentally evaluate it. We make the source code of our solver available.

D. Relation to Vacuity

Conceptually, under the frequently legitimate assumption that a system description can be translated into an LTL formula, our results extend to vacuity for LTL [GC04,Bee+01, KV03,Arm+03,Sim+10,Fis+08,Kup06]. Due to space constraints we refer to App. D for details.

E. Related Work

In [Cim+07] Cimatti et al. perform extraction of UCs for PSL to accelerate a PSL satisfiability solver by performing Boolean abstraction. Their notion of UCs is coarser than ours and their solver is based on BDDs and on SAT. An investigation of notions of UCs for LTL including the relation between UCs and vacuity is performed in [Sch12]. No implementation or experimental results are reported, and TR is not considered. Hantry et al. suggest a method to extract UCs for LTL in a tableau-based solver [HH11]. No implementation or experiments are reported. Awad et al. [Awa+12] use tableaux to extract UCs in the context of synthesizing business process templates. The description of the method is sketchy and incomplete, the notion of UC appears to be one of a subset of a set of formulas, and no detailed experimental evaluation is carried out. In [HSH12] the decision and search problems for minimal UCs for LTL are shown to be PSPACE- and FPSPACE-complete, respectively. In [CMT11] Cimatti et al. show how to prove and explain unfeasibility of message sequence charts for networks of hybrid automata. They consider a different specification language and use an SMT-based algorithm. Some work deals with unrealizable rather than unsatisfiable cores. [Cim+08] handles specifications in GR(1), which is a proper subset of LTL. Könighofer et al. present methods to help debugging unrealizable specifications by extracting unrealizable cores and simulating counterstrategies [KHB09] as well as performing error localization using model-based diagnosis [KHB10]. Raman and Kress-Gazit [RKG11] present a tool that points out unrealizable cores in the context of robot control. [Sch12] explores more fine-grained notions of unrealizable cores than [Cim+08,KHB09]. In vacuity Simmonds et al. [Sim+10] use SAT-based bounded model checking (e.g., [Bie+99,Bie09]) for vacuity detection. They only consider k-step vacuity, i.e., taking into account bounded model checking runs up to a bound k, and leave the problem of removing the bound k open. For a more extensive discussion on the relation between vacuity and UCs for LTL we refer to App. D and [Sch12].

$$
\begin{array}{lcl}
(\pi, i) \models 1, \not\models 0 & & \\
(\pi, i) \models p & \Leftrightarrow & p \in \pi[i] \\
(\pi, i) \models \neg\psi & \Leftrightarrow & (\pi, i) \not\models \psi \\
(\pi, i) \models \mathbf{X}\psi & \Leftrightarrow & (\pi, i+1) \models \psi \\
(\pi, i) \models \psi \vee \psi' & \Leftrightarrow & (\pi, i) \models \psi \text{ or } (\pi, i) \models \psi' \\
(\pi, i) \models \psi \wedge \psi' & \Leftrightarrow & (\pi, i) \models \psi \text{ and } (\pi, i) \models \psi' \\
(\pi, i) \models \psi \mathbf{U} \psi' & \Leftrightarrow & \exists i' \geq i \,.\, ((\pi, i') \models \psi' \wedge \forall i \leq i'' < i' \,.\, (\pi, i'') \models \psi) \\
(\pi, i) \models \psi \mathbf{R} \psi' & \Leftrightarrow & \forall i' \geq i \,.\, ((\pi, i') \models \psi' \vee \exists i \leq i'' < i' \,.\, (\pi, i'') \models \psi) \\
(\pi, i) \models \mathbf{F}\psi & \Leftrightarrow & \exists i' \geq i \,.\, (\pi, i') \models \psi \\
(\pi, i) \models \mathbf{G}\psi & \Leftrightarrow & \forall i' \geq i \,.\, (\pi, i') \models \psi
\end{array}
$$

Fig. 1. Semantics of LTL. π is a word in $(2^{AP})^\omega$, i is a time point in $\mathbb{N}$. π satisfies ϕ iff $(\pi, 0) \models \phi$.

F. Structure of the Paper

Section II starts with preliminaries. TR and its clausal normal form SNF are introduced in Sec. III. In Sec. IV we describe the construction of a resolution graph and its use to obtain a UC. The UCs obtained in Sec. IV are lifted from SNF to LTL in Sec. V and minimized in Sec. VI. In Sec. VII we provide examples that illustrate why these UCs are useful and how to obtain them. We discuss our implementation and experimental evaluation in Sec. VIII. Section IX concludes. Due to space constraints some proofs are sketched or omitted in the main part; these can be found in the appendices. For our implementation, examples, and log files see [www].

II. Preliminaries

We use a standard version of LTL, see, e.g., [Eme90]. Let $\mathbb{B}$ be the set of Booleans, and let AP be a finite set of atomic propositions. The set of *LTL formulas* is constructed inductively as follows. The Boolean constants 0 (false), 1 (true) $\in \mathbb{B}$ and any atomic proposition $p \in AP$ are LTL formulas. If ψ, ψ' are LTL formulas, so are $\neg\psi$ (not), $\psi \vee \psi'$ (or), $\psi \wedge \psi'$ (and), $\mathbf{X}\psi$ (next time), $\psi\mathbf{U}\psi'$ (until), $\psi\mathbf{R}\psi'$ (releases), $\mathbf{F}\psi$ (finally), and $\mathbf{G}\psi$ (globally). We use $\psi \rightarrow \psi'$ (implies) as an abbreviation for $\neg\psi \vee \psi'$. For the semantics of LTL see Fig. 1. An occurrence of a subformula ψ of an LTL formula ϕ has *positive polarity* (+) if it appears under an even number of negations in ϕ and *negative polarity* (−) otherwise.

Input: A set of SNF clauses C.
Output: *Unsat* if C is unsatisfiable; *sat* otherwise.

```
1  M ← C; if □ ∈ M then return unsat;
2  saturate(M); if □ ∈ M then return unsat;
3  augment(M);
4  saturate(M); if □ ∈ M then return unsat;
5  M' ← ∅;
6  while M' ≠ M do
7      M' ← M;
8      for c ∈ C . c is an eventuality clause do
9          C' ← {□};
10         repeat
11             initialize-BFS-loop-search-iteration(M, c, C', L);
12             saturate-step-xx(L);
13             C' ← {c' ∈ L | c' has empty X part};
14             C'' ← {(G(Q)) | (G((0) ∨ (X(Q ∨ l)))) ∈
                L generated by [BFS-loop-it-init-c]};
15             found ← subsumes(C', C'');
16         until found or C' = ∅ ;
17         if found then
18             derive-BFS-loop-search-conclusions(c, C', M);
19             saturate(M); if □ ∈ M then return unsat;
20 return sat;
```

Fig. 2. LTL satisfiability checking via TR in TRP++.

III. Temporal Resolution (TR)

In this section we describe TR [FDP01] as implemented in TRP++ [HK03,HK04,trp++]. We first explain the clausal normal form TR is based on. Then we provide a concise description of TR as required for the purposes of this paper.

A. Separated Normal Form (SNF)

TR works on formulas in a clausal normal form called separated normal form (SNF) [Fis91,FN92,FDP01]. For any atomic proposition $p \in AP$ p and $\neg p$ are *literals*. Let $p_1, \ldots, p_n$, $q_1, \ldots, q_{n'}$, l with $0 \leq n, n'$ be literals such that $p_1, \ldots, p_n$ and $q_1, \ldots, q_{n'}$ are pairwise different. Then (i) $(p_1 \vee \ldots \vee p_n)$ is an *initial clause*; (ii) $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{X}(q_1 \vee \ldots \vee q_{n'}))))$ is a *global clause*; and (iii) $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{F}(l))))$ is an *eventuality clause*. l is called an *eventuality literal*. As usual an empty disjunction (resp. conjunction) stands for 0 (resp. 1). $()$ or $(\mathbf{G}())$, denoted $\Box$, stand for 0 or $\mathbf{G}(0)$ and are called *empty clause*. The set of all SNF clauses is denoted $\mathbb{C}$. Let $c_1, \ldots, c_n$ with $0 \leq n$ be SNF clauses. Then $\bigwedge_{1 \leq i \leq n} c_i$ is an LTL formula in *SNF*. Every LTL formula ϕ can be transformed into an equisatisfiable formula ϕ' in SNF [FDP01].

B. TR in TRP++

The production rules of TRP++ are shown in Tab. I. The second column assigns a name to a production rule. The third and fifth columns list the premises. The seventh column gives the conclusion. Columns 4, 6, and 8 are described below. Columns 9–11 become relevant only in later sections.

The algorithm in Fig. 2 provides a high level view of TR in TRP++ [HK04]. The algorithm takes a set of starting clauses C in SNF as input. It returns *unsat* if C is found to be unsatisfiable (by deriving $\Box$) and *sat* otherwise. Resolution between two initial or two global clauses or between an initial and a global clause is performed by a straightforward extension of propositional resolution (e.g., [Rob65,FM09,BG01]). The corresponding production rules are listed next to *saturation* in Tab. I. Given a set of SNF clauses C we say that one *saturates* C if one applies these production rules to clauses in C until no new clauses are generated. Resolution between a set of initial and global clauses and an eventuality clause with eventuality literal l requires finding a set of global clauses that allows to infer conditions under which $\mathbf{XG}\neg l$ holds. Such a set of clauses is called a *loop* in $\neg l$. Loop search involves all production rules in Tab. I except [init-ii], [init-in], [step-nn], and [step-nx].

In line 1 the algorithm in Fig. 2 initializes M with the set of starting clauses and terminates iff one of these is the empty clause. Then, in line 2, it saturates M (terminating iff the empty clause is generated). In line 3 it *augments* M by applying production rule [aug1] to each eventuality clause in M and [aug2] once per eventuality literal in M, where wl is a fresh proposition. This is followed by another round of saturation in line 4. From now on the algorithm in Fig. 2 alternates between searching for a loop for some eventuality clause c (lines 9–18) and saturating M if loop search has generated new clauses (line 19). It terminates, if either the empty clause was derived (line 19) or if no new clauses were generated (line 20).

Loop search for some eventuality clause c may take several *iterations* (lines 11–15). Each loop search iteration uses saturation restricted to [step-xx] as a subroutine (line 12). Therefore, each loop search iteration has its own set of clauses L in which it works. We call M and L *partitions*. Columns 4, 6, and 8 in Tab. I indicate whether a premise (resp. conclusion) of a production rule is taken from (resp. put into) the main partition (M), the loop partition of the current loop search iteration (L), the loop partition of the previous loop search iteration (L'), or either of M or L as long as premises and conclusion are in the same partition (ML). In line 11 partition L of a loop search iteration is initialized by applying production rule [BFS-loop-it-init-x] once to each global clause with non-empty $\mathbf{X}$ part in M, rule [BFS-loop-it-init-n] once to each global clause with empty $\mathbf{X}$ part in M, and rule [BFS-loop-it-init-c] once to each global clause with empty $\mathbf{X}$ part in the partition of the previous loop search iteration L'. Notice that by construction at this point L contains only global clauses with non-empty $\mathbf{X}$ part. Then L is saturated using only rule [step-xx] (line 12). A loop has been found iff each global clause with empty $\mathbf{X}$ part that was derived in the previous loop search iteration is subsumed by at least one global clause with empty $\mathbf{X}$ part that was derived in the current loop search iteration (lines 13–15). Subsumption between a pair of clauses corresponds to an instance of production rule [BFS-loop-it-sub]; note, though, that this rule does not produce a new clause but records a relation between two clauses to be used later for extraction of a UC. Loop search for c terminates, if either a loop has been found or no clauses with empty $\mathbf{X}$ part were derived (line 16). If a loop has been found, rules [BFS-loop-conclusion1] and [BFS-loop-conclusion2] are applied once to each global clause with empty $\mathbf{X}$ part that was derived in the current loop search iteration (line 18) to obtain the loop search conclusions for the main partition.

TABLE I

PRODUCTION RULES USED IN TRP++. LET $P \equiv p_1 \vee \ldots \vee p_n$, $Q \equiv q_1 \vee \ldots \vee q_{n'}$, $R \equiv r_1 \vee \ldots \vee r_{n''}$, AND $S \equiv s_1 \vee \ldots \vee s_{n'''}$.

	rule	premise 1	part.	premise 2	part.	conclusion	part.	p.1 – c	p.2 – c	vt. c
saturation	init-ii	$(P \vee l)$	M	$(\neg l \vee Q)$	M	$(P \vee Q)$	M	✔	✔	✔
	init-in	$(P \vee l)$	M	$(\mathbf{G}(\neg l \vee Q))$	M	$(P \vee Q)$	M	✔	✔	✔
	step-nn	$(\mathbf{G}(P \vee l))$	M	$(\mathbf{G}(\neg l \vee Q))$	M	$(\mathbf{G}(P \vee Q))$	M	✔	✔	✔
	step-nx	$(\mathbf{G}(P \vee l))$	M	$(\mathbf{G}((Q) \vee (\mathbf{X}(\neg l \vee R))))$	M	$(\mathbf{G}((Q) \vee (\mathbf{X}(P \vee R))))$	M	✔	✔	✔
	step-xx	$(\mathbf{G}((P) \vee (\mathbf{X}(Q \vee l))))$	ML	$(\mathbf{G}((R) \vee (\mathbf{X}(\neg l \vee S))))$	ML	$(\mathbf{G}((P \vee R) \vee (\mathbf{X}(Q \vee S))))$	ML	✔	✔	✔
augmentation	aug1	$(\mathbf{G}((P) \vee (\mathbf{F}(l))))$			M	$(\mathbf{G}(P \vee l \vee wl))$	M	✔	—	✔
	aug2	$(\mathbf{G}((P) \vee (\mathbf{F}(l))))$			M	$(\mathbf{G}((\neg wl) \vee (\mathbf{X}(l \vee wl))))$	M	✖	—	✔
BFS loop search	BFS-loop-it-init-x	$c \equiv (\mathbf{G}((P) \vee (\mathbf{X}(Q))))$ with $\lvert Q \rvert > 0$			M	c	L	✔	—	✔
	BFS-loop-it-init-n	$(\mathbf{G}(P))$			M	$(\mathbf{G}((0) \vee (\mathbf{X}(P))))$	L	✔	—	✔
	BFS-loop-it-init-c	$(\mathbf{G}(P))$	L'	$(\mathbf{G}((Q) \vee (\mathbf{F}(l))))$	M	$(\mathbf{G}((0) \vee (\mathbf{X}(P \vee l))))$	L	✖	✖	✔
	BFS-loop-it-sub	$c \equiv (\mathbf{G}(P))$ with $c \rightarrow (\mathbf{G}(Q))$			L	$(\mathbf{G}((0) \vee (\mathbf{X}(Q \vee l))))$ generated by BFS-loop-it-init-c	L	✔	—	✖
	BFS-loop-conclusion1	$(\mathbf{G}(P))$	L	$(\mathbf{G}((Q) \vee (\mathbf{F}(l))))$	M	$(\mathbf{G}(P \vee Q \vee l))$	M	✔	✔	✔
	BFS-loop-conclusion2	$(\mathbf{G}(P))$	L	$(\mathbf{G}((Q) \vee (\mathbf{F}(l))))$	M	$(\mathbf{G}((\neg wl) \vee (\mathbf{X}(P \vee l))))$	M	✔	✖	✔

IV. UC EXTRACTION

In this section we describe, given an unsatisfiable set of SNF clauses C, how to obtain a subset of C, C^{uc}, that is by itself unsatisfiable from an execution of the algorithm in Fig. 2. The general idea of the construction is unsurprising in that during the execution of the algorithm in Fig. 2 a resolution graph is built that records which clauses were used to generate other clauses (Def. 1). Then the resolution graph is traversed backwards from the empty clause to find the subset of C that was actually used to prove unsatisfiability (Def. 2). The main concern of Def. 1, 2, and their proof of correctness in Thm. 1 is therefore that/why certain parts of the TR proof do not need to be taken into account when determining C^{uc}. Remark 1 complements this by showing for other parts of the TR proof that they are indeed required to obtain C^{uc}. Finally, in Remark 2, the specifics of TR in the algorithm in Fig. 2 and of Def. 1, 2 are used to optimize construction of the resolution graph.

Definition 1 (Resolution Graph)**.** *A resolution graph G is a directed graph consisting of (i) a set of vertices V, (ii) a set of directed edges $E \subseteq V \times V$, (iii) a labeling of vertices with SNF clauses $L_V : V \rightarrow \mathbb{C}$, and (iv) a partitioning $\mathcal{Q}^V$ of the set of vertices V into one main partition M^V and one partition L_i^V for each BFS loop search iteration: $\mathcal{Q}^V : V = M^V \uplus L_0^V \uplus \ldots \uplus L_n^V$. Let C be a set of SNF clauses. During an execution of the algorithm in Fig. 2 with input C a resolution graph G is constructed as follows.*

In line 1 G is initialized: (i) V contains one vertex v per clause c in C: $V = \{v_c \mid c \in C\}$, (ii) E is empty: $E = \emptyset$, (iii) each vertex is labeled with the corresponding clause: $L_V : V \rightarrow C, L_V(v_c) = c$, and (iv) the partitioning $\mathcal{Q}^V$ contains only the main partition M^V, which contains all vertices: $\mathcal{Q}^V : M^V = V$.

Whenever a new BFS loop search iteration is entered (line 11), a new partition L_i^V is created and added to $\mathcal{Q}^V$. For each application of a production rule from Tab. I that either generates a new clause in partition M or L or is the first application of rule BFS-loop-it-sub to clause c'' in C'' in line 15: (i) if column 11 (vt. c) of Tab. I contains ✔, then a new vertex v is created for the conclusion c (which is a new clause), labeled with c, and put into partition M^V or L_i^V; (ii) if column 9 (p.1 – c) (resp. column 10 (p.2 – c)) contains ✔, then an edge is created from the vertex labeled with premise 1 (resp. premise 2) in partition M^V or L_i^V to the vertex labeled with the conclusion in partition M^V or L_i^V.

Definition 2 (UC in SNF)**.** *Let C be a set of SNF clauses to which the algorithm in Fig. 2 has been applied and shown unsatisfiability, let G be the resolution graph, and let $v_\square$ be the (unique) vertex in the main partition M^V of the resolution graph G labeled with the empty clause $\square$. Let G' be the smallest subgraph of G that contains $v_\square$ and all vertices in G (and the corresponding edges) that are backward reachable from $v_\square$. The* UC of C in SNF*, C^{uc}, is the subset of C such that there exists a vertex v in the subgraph G', labeled with $c \in C$, and contained in the main partition M^V of G: $C^{uc} = \{c \in C \mid \exists v \in V_{G'} \,.\, L_V(v) = c \wedge v \in M^V\}$.*

Theorem 1 (Unsatisfiability of UC in SNF)**.** *Let C be a set of SNF clauses to which the algorithm in Fig. 2 has been applied and shown unsatisfiability, and let C^{uc} be the UC of C in SNF. Then C^{uc} is unsatisfiable.*

Assume for a moment that in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I all ✖ are replaced with ✔, i.e., that each conclusion in the resolution graph is connected by an edge to each of its premises rather than only to a subset of them. In that case the UC in SNF according to Def. 2 would contain all clauses of the set of starting clauses C that contributed to deriving the empty clause and, hence, to establishing unsatisfiability of C. In that case it would follow directly from the correctness of TR that C^{uc} is unsatisfiable. It remains to show that not including an edge (i) from premise 1 to the conclusion for rule aug2, (ii) from premise 2 to the conclusion for rule BFS-loop-conclusion2, (iii) from premise 2 to the conclusion for rule BFS-loop-it-init-c, and (iv) from premise 1 to the conclusion for rule BFS-loop-it-init-c in the resolution graph G maintains the fact that the resulting C^{uc} is unsatisfiable.

To see the intuition behind (i) note that for a vertex v_c labeled with a conclusion c of rule aug2 in the main partition M^V to be backward reachable from the (unique) vertex in the main partition M^V of the resolution graph G labeled with the empty clause $\square$, $v_\square$, the occurrence of $\neg wl$ in c must be "resolved away" at some point on the path from v_c to $v_\square$. It turns out that this can only happen by resolution with a clause that is derived from the conclusion of rule aug1 applied to an eventuality clause c' with eventuality literal l. By construction of the resolution graph G $v_{c'}$ must be backward reachable from $v_\square$ and, therefore, c' must be included in the UC in SNF.

Hence, an execution of the algorithm in Fig. 2 with input C^{uc} will produce c from c'.

A similar reasoning as for (i) applies to (ii).

For (iii) note that a conclusion of rule BFS-loop-it-init-c can only be backward reachable from $v_\Box$ if the corresponding BFS loop search iteration is successful and a vertex labeled with one of the resulting conclusions of rules BFS-loop-conclusion1 or BFS-loop-conclusion2 is backward reachable from $v_\Box$. The latter fact implies that an eventuality clause with the same eventuality literal as in premise 2 of rule BFS-loop-it-init-c is present in the UC in SNF. Hence, an execution of the algorithm in Fig. 2 with input C^{uc} will produce premise 2 of BFS-loop-it-init-c as required.

Finally, (iv) is obtained by understanding that in a BFS loop search iteration the premises 1 of rule BFS-loop-it-init-c essentially constitute a hypothetical fixed point; if the BFS loop search iteration is successful, then the hypothetical fixed point is proven to be an actual fixed point. For the correctness of a proof of unsatisfiability of C it is only relevant that this hypothetical fixed point is shown to be an actual fixed point but not how the hypothesis is obtained.

For a formalization of the above reasoning see App. A.

By taking the fact that each vertex in the resolution graph has at most 2 incoming edges into account, the first part of the following Prop. 1 is immediate from Def. 1 and 2. The second part is obtained by bounding the number of (i) different clauses in each partition, (ii) iterations in each loop search by the length of the longest monotonically increasing sequence of Boolean formulas over AP, and (iii) loop searches by the number of different loop search conclusions.

Proposition 1 (Complexity of UC Extraction)**.** *Let C be a set of SNF clauses to which the algorithm in Fig. 2 is applied and shows unsatisfiability. Construction and backward traversal of the resolution graph and, hence, construction of C^{uc} according to Def. 2 can be performed in time $\mathcal{O}(|V|)$ in addition to the time required to run the algorithm in Fig. 2. $|V|$ is at most exponential in $|AP| + log(|C|)$.*

Remark 1 (Minimality of Set of Premises to Include in Resolution Graph)**.** *Theorem 1 shows that not including premises for production rules marked by ✖ in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I during the construction of the resolution graph still leads to a UC. It does not discuss whether the remaining premises, marked by ✔ in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I, actually need to be included to guarantee a UC. For all premises of all production rules marked by ✔ in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I it turns out that they are indeed required to obtain a UC. The proof in App. A is essentially obtained by providing suitable examples.*

Remark 2 (Pruning the Resolution Graph)**.** *The specifics of TR in the algorithm in Fig. 2 and the fact that not all premises need to be included during the construction of the resolution graph allow to optimize extraction of UCs by pruning the resolution graph during the execution of the algorithm in Fig. 2 extended with the construction in Def. 1, 2 as follows. (i) Notice that after the completion of a (successful or unsuccessful) loop search for some eventuality clause c in lines 9–19 of the algorithm in Fig. 2 no new edges between the main partition and one of the partitions used during the just completed loop search for c will be created. Hence, after completion of an execution of lines 9–19 of the algorithm in Fig. 2 vertices not backward reachable from the main partition can be pruned from the resolution graph. (ii) Moreover, note that, because there is no edge from instances of premise 1 to the conclusion induced by production rule BFS-loop-it-init-c, there are no outgoing edges from a failed loop search iteration (lines 11–15 of the algorithm in Fig. 2). Therefore, if a loop search iteration fails, all vertices and edges in the partition of that loop search iteration can be pruned from the resolution graph right away.*

V. From LTL to SNF and Back

We use a structure-preserving translation (e.g., [PG86]) to translate an LTL formula into a set of SNF clauses, which slightly differs from the translation suggested in [FDP01]. It is well known that ϕ and $SNF(\phi)$ according to Def. 3 are equisatisfiable and that a satisfying assignment for ϕ (resp. $SNF(\phi)$) can be extended (resp. restricted) to a satisfying assignment of $SNF(\phi)$ (resp. ϕ).

Definition 3 (Translation from LTL to SNF)**.** *Let ϕ be an LTL formula over atomic propositions AP, and let $X = \{x, x', \ldots\}$ be a set of fresh atomic propositions not in AP. Assign each occurrence of a subformula ψ in ϕ a Boolean value or a proposition according to column 2 of Tab. II, which is used to reference ψ in the SNF clauses for its superformula. Moreover, assign each occurrence of ψ a set of SNF clauses according to column 3 or 4 of Tab. II. Let $SNF_{aux}(\phi)$ be the set of all SNF clauses obtained from ϕ that way. Then the* SNF of ϕ *is defined as* $SNF(\phi) \equiv x_\phi \wedge \bigwedge_{c \in SNF_{aux}(\phi)} c$.

In the following Def. 4 we describe how to map a UC in SNF back to a UC in LTL. The main idea in its proof of correctness (Thm. 2) is to compare the SNF of ϕ and ϕ^{uc} by partitioning the SNF clauses into three sets: one that is shared by the two SNFs, one that replaces some occurrences of propositions in $SNF(\phi)$ with 1 or 0, and one whose clauses are only in $SNF(\phi)$. Then one can show that the UC of ϕ in SNF must be contained in the first partition.

Definition 4 (Mapping a UC in SNF to a UC in LTL)**.** *Let ϕ be an unsatisfiable LTL formula, let $SNF(\phi)$ be its SNF, and let C^{uc} be the UC of $SNF(\phi)$ in SNF. Then the* UC of ϕ in LTL, *ϕ^{uc}, is obtained as follows. For each positive (resp. negative) polarity occurrence of a proper subformula ψ of ϕ with proposition x_ψ according to Tab. II, replace ψ in ϕ with 1 (resp. 0) iff C^{uc} contains no clause with an occurrence of proposition x_ψ that is marked blue boxed in Tab. II. (We are sloppy in that we "replace" subformulas of replaced subformulas, while in effect they simply vanish.)*

Theorem 2 (Unsatisfiability of UC in LTL)**.** *Let ϕ be an unsatisfiable LTL formula, and let ϕ^{uc} be the UC of ϕ in*

TABLE II
TRANSLATION FROM LTL TO SNF.

Subf.	Prop.	SNF Clauses ($+$ polarity occurrences)	SNF Clauses ($-$ polarity occurrences)
$1/0/p$	$1/0/p$	—	—
$\neg\psi$	$x_{\neg\psi}$	$(\mathbf{G}(x_{\neg\psi} \rightarrow (\neg\boxed{x_{\psi}})))$	$(\mathbf{G}((\neg x_{\neg\psi}) \rightarrow \boxed{x_{\psi}}))$
$\psi \wedge \psi'$	$x_{\psi\wedge\psi'}$	$(\mathbf{G}(x_{\psi\wedge\psi'} \rightarrow \boxed{x_{\psi}}))$, $(\mathbf{G}(x_{\psi\wedge\psi'} \rightarrow \boxed{x_{\psi'}}))$	$(\mathbf{G}((\neg x_{\psi\wedge\psi'}) \rightarrow ((\neg\boxed{x_{\psi}}) \vee (\neg\boxed{x_{\psi'}}))))$
$\psi \vee \psi'$	$x_{\psi\vee\psi'}$	$(\mathbf{G}(x_{\psi\vee\psi'} \rightarrow (\boxed{x_{\psi}} \vee \boxed{x_{\psi'}})))$	$(\mathbf{G}((\neg x_{\psi\vee\psi'}) \rightarrow (\neg\boxed{x_{\psi}})))$, $(\mathbf{G}((\neg x_{\psi\vee\psi'}) \rightarrow (\neg\boxed{x_{\psi'}})))$
$\mathbf{X}\psi$	$x_{\mathbf{X}\psi}$	$(\mathbf{G}(x_{\mathbf{X}\psi} \rightarrow (\mathbf{X}\boxed{x_{\psi}})))$	$(\mathbf{G}((\neg x_{\mathbf{X}\psi}) \rightarrow (\mathbf{X}\neg\boxed{x_{\psi}})))$
$\mathbf{G}\psi$	$x_{\mathbf{G}\psi}$	$(\mathbf{G}(x_{\mathbf{G}\psi} \rightarrow (\mathbf{X}x_{\mathbf{G}\psi})))$, $(\mathbf{G}(x_{\mathbf{G}\psi} \rightarrow \boxed{x_{\psi}}))$	$(\mathbf{G}((\neg x_{\mathbf{G}\psi}) \rightarrow (\mathbf{F}\neg\boxed{x_{\psi}})))$
$\mathbf{F}\psi$	$x_{\mathbf{F}\psi}$	$(\mathbf{G}(x_{\mathbf{F}\psi} \rightarrow (\mathbf{F}\boxed{x_{\psi}})))$	$(\mathbf{G}((\neg x_{\mathbf{F}\psi}) \rightarrow (\mathbf{X}\neg x_{\mathbf{F}\psi})))$, $(\mathbf{G}((\neg x_{\mathbf{F}\psi}) \rightarrow (\neg\boxed{x_{\psi}})))$
$\psi\mathbf{U}\psi'$	$x_{\psi\mathbf{U}\psi'}$	$(\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (\boxed{x_{\psi'}} \vee \boxed{x_{\psi}})))$, $(\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (\boxed{x_{\psi'}} \vee (\mathbf{X}x_{\psi\mathbf{U}\psi'}))))$, $(\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (\mathbf{F}\boxed{x_{\psi'}})))$	$(\mathbf{G}((\neg x_{\psi\mathbf{U}\psi'}) \rightarrow (\neg\boxed{x_{\psi'}})))$, $(\mathbf{G}((\neg x_{\psi\mathbf{U}\psi'}) \rightarrow ((\neg\boxed{x_{\psi}}) \vee (\mathbf{X}\neg x_{\psi\mathbf{U}\psi'}))))$
$\psi\mathbf{R}\psi'$	$x_{\psi\mathbf{R}\psi'}$	$(\mathbf{G}(x_{\psi\mathbf{R}\psi'} \rightarrow \boxed{x_{\psi'}}))$, $(\mathbf{G}(x_{\psi\mathbf{R}\psi'} \rightarrow (\boxed{x_{\psi}} \vee (\mathbf{X}x_{\psi\mathbf{R}\psi'}))))$	$(\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow ((\neg\boxed{x_{\psi'}}) \vee (\neg\boxed{x_{\psi}}))))$, $(\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow ((\neg\boxed{x_{\psi'}}) \vee (\mathbf{X}\neg x_{\psi\mathbf{R}\psi'}))))$, $(\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow (\mathbf{F}\neg\boxed{x_{\psi'}})))$

LTL. Then ϕ^{uc} is unsatisfiable.

Remark 3 (Def. 4 UC is [Sch12] Def. 10 UC)**.** *In Def. 10 of [Sch12] a UC of an unsatisfiable formula in LTL is obtained by replacing some occurrences of positive polarity subformulas with* 1 *and some occurrences of negative polarity subformulas with* 0 *while maintaining unsatisfiability. By construction in Def. 4 and with Thm. 2 it is immediate that a UC in LTL according to Def. 4 above is a UC according to Def. 10 of [Sch12].*

VI. MINIMAL UCS

In this section we introduce notions of and algorithms to obtain minimal UCs. The results are either straightforward (Remark 4) or well known (Def. 5, Remark 5). Still, the material is needed in the experimental evaluation and within the flow of the paper this seems to be the appropriate place.

Definition 5 (Minimal UC in SNF and LTL)**.** *(See, e.g., [Sch12]: irreducible UC) A UC C^{uc} in SNF is* minimal *iff $\forall c \in C^{uc}\ .\ C^{uc} \setminus \{c\}$ is satisfiable. A UC ϕ^{uc} in LTL is* minimal *iff there is no positive polarity occurrence of a subformula that can be replaced with* 1 *and no negative polarity occurrence of a subformula that can be replaced with* 0 *without making ϕ^{uc} satisfiable.*

Remark 4 (Minimal UC in SNF No Guarantee for Minimal UC in LTL)**.** *Let ϕ be an unsatisfiable LTL formula, C its translation to SNF, C^{uc} a minimal UC of C in SNF, and ϕ^{uc} the UC of ϕ in LTL obtained by mapping C^{uc} back to LTL via Def. 4. Then ϕ^{uc} is not necessarily minimal.*

Remark 5 (Extraction of Minimal UCs)**.** *A common way to obtain minimal UCs works by repeatedly attempting to remove parts of a UC (e.g., [CD91,Bak+93,ZM03a,MS10]). If the modified formula is still unsatisfiable, then the removal is made permanent; otherwise the removal is undone. The procedure continues until all parts of the UC have been considered for removal. This is called* deletion-based extraction of minimal UCs *(e.g., [CD91,MS10]). In the case of LTL the algorithm attempts to replace positive polarity occurrences of subformulas with* 1 *and negative polarity ones with* 0*. It terminates, if no more replacements can be performed without making the resulting formula satisfiable. Naturally, this method may be expensive due to the number of satisfiability tests to be performed. It is therefore often used to minimize a UC that has been obtained by other means such as those described in Sec. IV, V (see, e.g., [CD91,Bak+93,ZM03a, MS10]). Potential optimizations of the minimization algorithm include binary search (e.g., [ZH02,Jun01,MS10,KHB09]) and reusing intermediate results (e.g., [KHB09]).*

VII. EXAMPLES

In this section we first present examples of using UCs for LTL to help understanding a specification given in LTL. Then we show an example of TR with the corresponding resolution graph and UC extraction in SNF. Except for minor rewriting, all UCs in this section were obtained with our implementation.

A. Using UCs in LTL to Help Understanding LTL Specifications

We start with a toy example and then proceed to a more realistic one. The first example (1a)–(1c) is based on [JB06]. We would like to see whether a req (request) can be issued (1d). This is impossible, as (1a) requires a req to be followed by 3 gnts (grant), whereas (1b) forbids two subsequent gnts. The UC in (2) clearly shows this.

$$\begin{aligned} &(\mathbf{G}(req \rightarrow ((\mathbf{X}gnt) \wedge (\mathbf{XX}gnt) \wedge (\mathbf{XXX}gnt)))) && \text{(1a)}\\ \wedge\ &(\mathbf{G}(gnt \rightarrow \mathbf{X}\neg gnt)) && \text{(1b)}\\ \wedge\ &(\mathbf{G}(cancel \rightarrow \mathbf{X}((\neg gnt)\mathbf{U}go))) && \text{(1c)}\\ \wedge\ &(\mathbf{F}req) && \text{(1d)} \end{aligned}$$

$$(\mathbf{G}(req \rightarrow ((\mathbf{X}gnt) \wedge (\mathbf{XX}gnt)))) \wedge (\mathbf{G}(gnt \rightarrow \mathbf{X}\neg gnt)) \wedge (\mathbf{F}req) \quad (2)$$

The second example (3) in Fig. 3 is adapted from a lift specification in [Har05] (we used a somewhat similar example in [Sch12]). The lift has two floors, indicated by f_0 and f_1. On each floor there is a button to call the lift (b_0, b_1). sb is 1 if some button is pressed. If the lift moves up, then up must be 1; if it moves down, then up must be 0. u switches turns between actions by users of the lift (u is 1) and actions by the lift (u is 0). For more details we refer to [Har05].

We first assume that an engineer is interested in seeing whether it is possible that b_1 is continuously pressed (4). As the UC (5) shows, this is impossible as b_1 must be 0 at the beginning.

$$\mathbf{G}b_1 \quad (4) \qquad\qquad (\neg b_1) \wedge \mathbf{G}b_1 \quad (5)$$

Now the engineer modifies her query such that b_1 is pressed only from time point 1 on (6). As shown by the UC in (7)

$$
\begin{aligned}
&(\neg u) \wedge (f_0) \wedge (\neg b_0) \wedge (\neg b_1) \wedge (\neg up) && \text{(3a)}\\
\wedge\ &(\mathbf{G}((u \rightarrow \neg \mathbf{X} u) \wedge ((\neg \mathbf{X} u) \rightarrow u))) && \text{(3b)}\\
\wedge\ &(\mathbf{G}(f_0 \rightarrow \neg f_1)) && \text{(3c)}\\
\wedge\ &(\mathbf{G}((f_0 \rightarrow \mathbf{X}(f_0 \vee f_1)) \wedge (f_1 \rightarrow \mathbf{X}(f_0 \vee f_1)))) && \text{(3d)}\\
\wedge\ &(\mathbf{G}(u \rightarrow ((f_0 \rightarrow \mathbf{X} f_0) \wedge ((\mathbf{X} f_0) \rightarrow f_0)))) && \text{(3e)}\\
\wedge\ &(\mathbf{G}(u \rightarrow ((f_1 \rightarrow \mathbf{X} f_1) \wedge ((\mathbf{X} f_1) \rightarrow f_1)))) && \text{(3f)}\\
\wedge\ &(\mathbf{G}(((\neg u) \rightarrow ((b_0 \rightarrow \mathbf{X} b_0) \wedge ((\mathbf{X} b_0) \rightarrow b_0))))) && \text{(3g)}\\
\wedge\ &(\mathbf{G}(((\neg u) \rightarrow ((b_1 \rightarrow \mathbf{X} b_1) \wedge ((\mathbf{X} b_1) \rightarrow b_1))))) && \text{(3h)}\\
\wedge\ &(\mathbf{G}(((b_0 \wedge \neg f_0) \rightarrow \mathbf{X} b_0) \wedge ((b_1 \wedge \neg f_1) \rightarrow \mathbf{X} b_1))) && \text{(3i)}\\
\wedge\ &(\mathbf{G}((f_0 \wedge \mathbf{X} f_0) \rightarrow ((up \rightarrow \mathbf{X} up) \wedge ((\mathbf{X} up) \rightarrow up)))) && \text{(3j)}\\
\wedge\ &(\mathbf{G}((f_1 \wedge \mathbf{X} f_1) \rightarrow ((up \rightarrow \mathbf{X} up) \wedge ((\mathbf{X} up) \rightarrow up)))) && \text{(3k)}\\
\wedge\ &(\mathbf{G}(((f_0 \wedge \mathbf{X} f_1) \rightarrow up) \wedge ((f_1 \wedge \mathbf{X} f_0) \rightarrow \neg up))) && \text{(3l)}\\
\wedge\ &(\mathbf{G}((sb \rightarrow (b_0 \vee b_1)) \wedge ((b_0 \vee b_1) \rightarrow sb))) && \text{(3m)}\\
\wedge\ &(\mathbf{G}(((f_0 \wedge \neg sb) \rightarrow (f_0 \mathbf{U} (sb \mathbf{R} ((\mathbf{F} f_0) \wedge (\neg up)))))) && \text{(3n)}\\
\wedge\ &(\mathbf{G}(((f_1 \wedge \neg sb) \rightarrow (f_1 \mathbf{U} (sb \mathbf{R} ((\mathbf{F} f_0) \wedge (\neg up)))))) && \text{(3o)}\\
\wedge\ &(\mathbf{G}((b_0 \rightarrow \mathbf{F} f_0) \wedge (b_1 \rightarrow \mathbf{F} f_1))) && \text{(3p)}
\end{aligned}
$$

Fig. 3. A lift specification.

that turns out to be impossible, too.

$$\mathbf{X}\mathbf{G} b_1 \tag{6}$$

$$(\neg u) \wedge ((\neg b_1) \wedge ((\mathbf{G}((\neg u) \rightarrow ((\mathbf{X} b_1) \rightarrow b_1))) \wedge (\mathbf{X}\mathbf{G} b_1))) \tag{7}$$

The engineer now tries to have b_1 pressed from time point 2 on and, again, obtains a UC. She becomes suspicious and checks whether b_1 can be pressed at all (8). She sees that b_1 cannot be pressed and, therefore, this specification of a lift must contain a bug. She can now use the UC in (9a)–(9f) to track down the problem. This example illustrates the use of UCs for debugging, as (9a)–(9f) is significantly smaller than (3).

$$\mathbf{F} b_1 \tag{8}$$

$$
\begin{aligned}
&(f_0) \wedge (\neg b_1) \wedge (\neg up) && \text{(9a)}\\
\wedge\ &(\mathbf{G}(f_0 \rightarrow \neg f_1)) && \text{(9b)}\\
\wedge\ &(\mathbf{G}(f_0 \rightarrow \mathbf{X}(f_0 \vee f_1))) && \text{(9c)}\\
\wedge\ &(\mathbf{G}((f_0 \wedge \mathbf{X} f_0) \rightarrow ((\mathbf{X} up) \rightarrow up))) && \text{(9d)}\\
\wedge\ &(\mathbf{G}((f_0 \wedge \mathbf{X} f_1) \rightarrow up)) && \text{(9e)}\\
\wedge\ &(\mathbf{G}(b_1 \rightarrow \mathbf{F} f_1)) && \text{(9f)}\\
\wedge\ &(\mathbf{F}(b_1)) && \text{(9g)}
\end{aligned}
$$

B. TR, Resolution Graph, and UC Extraction

In Fig. 4 we show an example of an execution of the TR algorithm with the corresponding resolution graph and UC extraction in SNF. The set of starting clauses C to be solved is $\mathbf{G}(a \vee \neg b)$, $\mathbf{G}(a \vee b \vee \mathbf{X}(a \vee b))$, $\mathbf{G}((\neg a) \vee \mathbf{X} a)$, $\mathbf{G}((\neg a) \vee \mathbf{F} \neg a)$, shown in the first row from the bottom in the rectangle shaded in light red. In Fig. 4 TR generally proceeds from bottom to top; in the top right corner the empty clause $\square$ is generated, indicating unsatisfiability. Clauses are connected with directed edges from premises to conclusions according to columns 9, 10 in Tab. I. Edges are labeled with production rules, where "BFS-loop" is abbreviated to "loop", "init" to "i", and "conclusion" to "conc". Saturation in line 2 of the algorithm in Fig. 2 produces $\mathbf{G}(a \vee b \vee \mathbf{X} a)$ in the second row from the bottom.[1] The other 2 clauses in that row are generated by augmentation (line 3). The following saturation (line 4) produces no new clauses. The dark green shaded rectangle is the loop partition for the first loop search iteration. Row 3 contains the clauses obtained by initialization of the BFS loop search iteration (line 11). Note that clause $\mathbf{G}(\mathbf{X} \neg a)$, generated by BFS-loop-it-init-c, has no edge coming in from the main partition. Row 4 then contains the clauses generated from those in row 3 by saturation restricted to step-xx (line 12). The subsumption test fails in this iteration, as none of the clauses in row 4 subsumes the empty clause (lines 13–15). The light green shaded rectangle is the loop partition for the second loop search iteration. Row 5 contains the clauses obtained by initialization and row 6 those obtained from them by restricted saturation. This time the subsumption test succeeds, and the loop search conclusions are shown in row 7 (line 18). Finally, row 8 contains the derivation of the empty clause $\square$ via saturation (line 19). The thick, dotted, blue clauses and edges show the part of the resolution graph that is backward reachable from $\square$. As all starting clauses in C are backward reachable from $\square$, the UC of C in SNF is C (note that this example serves to illustrate the mechanism rather than the benefit of UC extraction).

For a complete example that includes translation between LTL and SNF and leads to a proper UC see App. E.

[1] While it may seem that some clauses are not considered for loop initialization or saturation, this is due to either subsumption of one clause by another (e.g., $\mathbf{G}(a \vee b \vee \mathbf{X}(a \vee b))$ by $\mathbf{G}(a \vee b \vee \mathbf{X} a)$) or the fact that TRP++ uses *ordered* resolution (e.g., $\mathbf{G}(a \vee b \vee \mathbf{X} a)$ with $\mathbf{G}(\neg w \overline{a} \vee \mathbf{X}((\neg a) \vee w \overline{a}))$; [HK03,BG01]). Both are issues of completeness of TR and, therefore, not discussed in this paper.

VIII. Experimental Evaluation

Our implementation, examples, and log files are available from [www].

A. Implementation

We implemented extraction of UCs as described in Sec. IV, V in TRP++. We also implemented deletion-based minimization of UCs obtained with the previous method (Sec. VI). TRP++ provides a translation from LTL to SNF via an external tool. To facilitate tracing a UC in SNF back to the input formula in LTL we implemented a translator from LTL to SNF inside TRP++, which reimplements ideas from the external translator. We used parts of TSPASS [LH10] for our implementation. For data structures we used C++ Standard Library containers (e.g., [SL95,Jos12]), for graph operations the Boost Graph Library [bgl,SLL02].

B. Benchmarks

Our examples are based on [SD11]. In categories **crafted** and **random** and in family **forobots** we considered all unsatisfiable instances from [SD11]. The version of **alaska_lift** used here contains a small bug fix: in [DW+08,SD11] the subformula $\mathbf{X} u$ was erroneously written as literal Xu. Combining 2 variants of **alaska_lift** with 3 different scenarios we obtain 6 subfamilies of **alaska_lift**. For **anzu_genbuf** we invented 3 scenarios to obtain 3 subfamilies. For all benchmark families that consist of a sequence of instances of increasing difficulty we stopped after two instances that could not be solved due to time or memory out. Some instances were simplified to 0 during the translation from LTL to SNF; these instances were discarded. In Tab. III we give an overview of the benchmark families. Columns 1–3 give the category, name, and the source of the family. Columns 4–6 list the numbers of instances that were solved by our implementation without UC extraction,

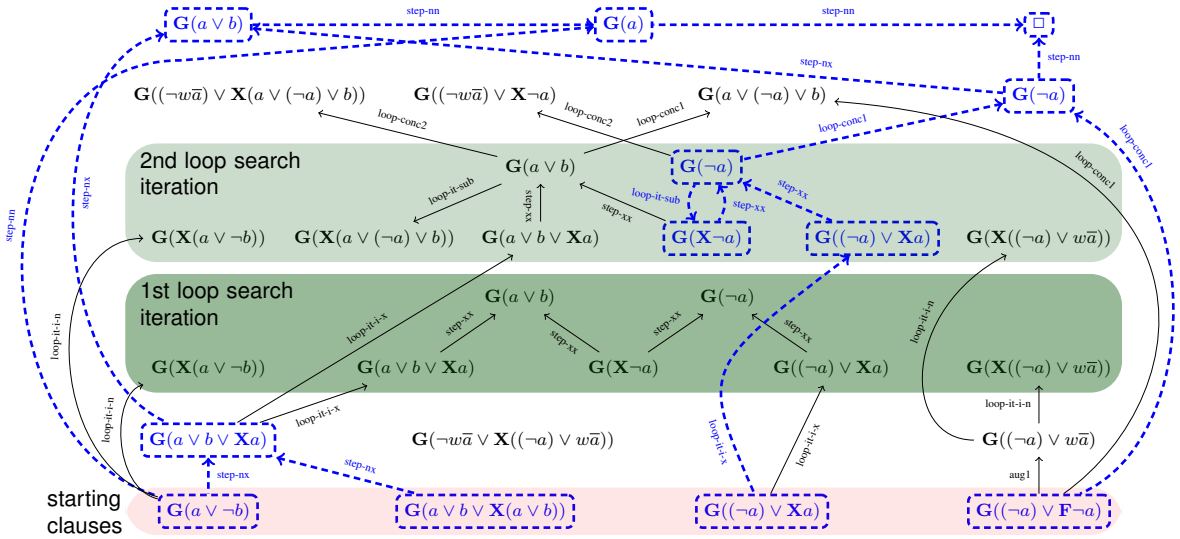


Fig. 4. Example of an execution of the TR algorithm with corresponding resolution graph and UC extraction in SNF.

TABLE III
OVERVIEW OF BENCHMARK FAMILIES.

category	family	source	# s. no UC	# s. UC	# s. minimal UC	|largest solved|
application	**alaska_lift**	[Har05,DW+08]	75	72	72	4605
	anzu_genbuf	[Blo+07]	16	16	16	1924
	forobots	[BDF09]	25	25	25	635
crafted	**schuppan_O1formula**	[SD11]	27	27	27	4006
	schuppan_O2formula	[SD11]	8	8	8	91
	schuppan_phltl	[SD11]	4	4	4	125
random	**rozier_random**	[RV10]	62	62	62	157
	trp	[HS02]	397	397	330	1421

with UC extraction, and with minimal UC extraction. Column 7 indicates the size (number of nodes in the syntax tree) of the largest instance solved without UC extraction.

C. Setup

The experiments were performed on a laptop with Intel Core i7 M 620 processor at 2 GHz running Ubuntu 12.04. Run time and memory usage were measured with `run` [run]. The time and memory limits were 600 seconds and 6 GB.

D. Extraction of UCs

In Fig. 5 (a), (b) we show the overhead that is incurred by extracting (non-minimal) UCs as described in Sec. IV, V over not extracting UCs. In Fig. 5 (c) we compare the sizes of the input formulas with the sizes of their (non-minimal) UCs. Our data show that extraction of UCs is possible with quite acceptable overhead in run time and memory usage. An analysis by category (plots see App. F) shows that the run time (resp. memory) overhead for instances of the **application** category, except for 3 that time out, is at most 100 % (resp. 300 %). Out of the 746 instances of all categories we considered with UC extraction disabled, 48 were simplified to 0 in the translation to SNF, 614 were shown to be unsatisfiable by TR, and 84 remained unsolved. Enabling UC extraction results in 3 time or memory outs out of 614 instances. The resulting UCs are often significantly smaller than the input formula. Separate plots by category (see App. F) indicate that instances of the **application** category are reduced comparatively well.

E. Extraction of Minimal UCs

Figure 5 (d)–(f) show the costs and benefits of applying deletion-based minimization (Sec. VI) to (non-minimal) UCs obtained as described in Sec. IV, V. Costs and benefits are somewhat varied. Minimal UCs can be computed for all instances for which (non-minimal) UCs were obtained except for all 67 instances in family **trp_N12y**. A closer analysis shows that most instances with high benefits are in the **random** category; some are also in family **forobots**. The largest reduction seen is 62 % from 469 to 177 nodes in the syntax tree of an instance from family **pltl_12x**. In the **application** category several instances exhibit reductions in the range between 20 % and 30 %. On several instances it takes more than 10 times as long to obtain a minimal UC as it takes to obtain a (non-minimal) UC. All instances that take more than 25 % longer are from the **forobots** family, the **schuppan_phltl** family, or the **random** category.

F. Optimizations

In Fig. 5 (g)–(l) we show the benefit of the optimizations described in Sec. IV when extracting (non-minimal) UCs. We show the impact on the peak size of the resolution graph rather than on run time or memory, as the former is implementation independent. We note that in several instances the overhead of disabling all optimizations in terms of run time and memory is larger than 100 %. The impact of including premise 1 of BFS-loop-it-init-c during construction of the resolution graph and disabling immediate pruning of vertices and edges in partitions of failed loop search iterations from the resolution graph in Fig. 5 (h) (the former implies the latter) and of disabling pruning non-reachable vertices from the resolution graph between loop searches in Fig. 5 (k) is quite significant. The impact in the remaining cases (Fig. 5 (g), (i), (j)) is negligible. However, in cases (i) and (j) there is an instance where disabling the optimization leads to a larger (non-minimal) UC. This occurs more often also in case (h).

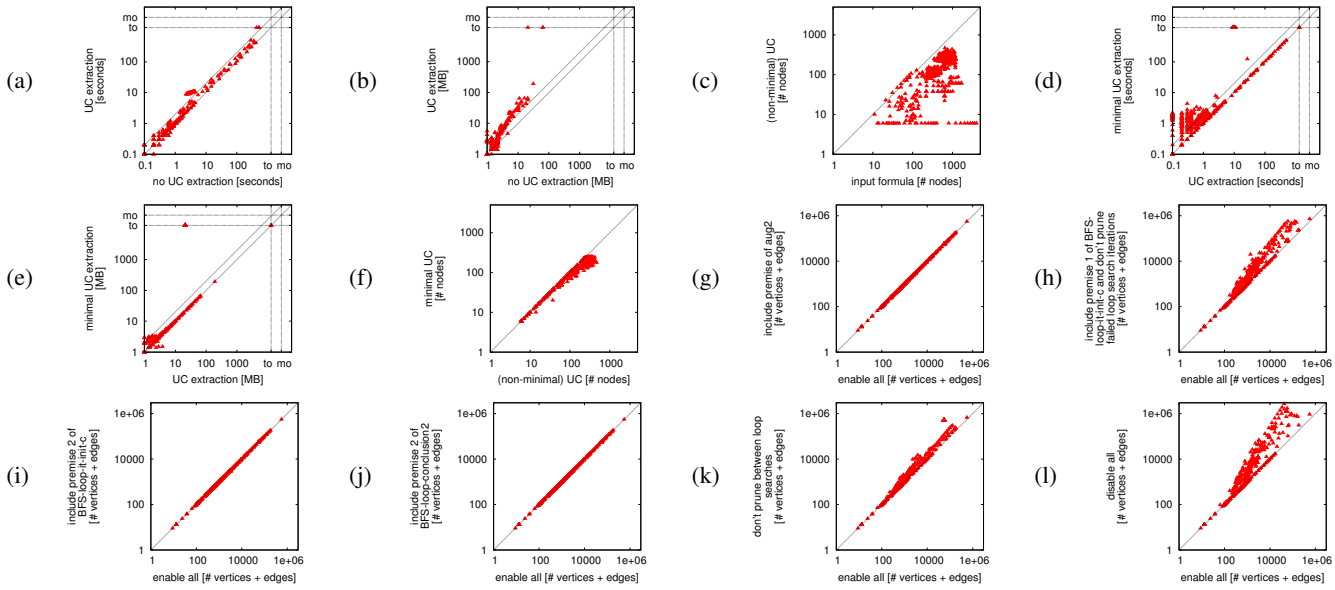


Fig. 5. (a)–(c) compare UC extraction (y-axis) with no UC extraction (x-axis). (a) and (b) show the overhead incurred in terms of run time (in seconds) and memory (in MB). (c) shows the size reduction obtained, where size is measured as the number of nodes in the syntax trees. (d)–(f) compare minimal UC extraction (y-axis) with UC extraction (x-axis). (d) and (e) show overhead incurred in terms of run time (in seconds) and memory (in MB). (f) shows the size reduction obtained, where size is measured as the number of nodes in the syntax trees. The off-center diagonal in (a), (b), (d), and (e) shows where $y = 2x$. (g)–(l) show the benefit of optimizations as reduction in peak size of resolution graph (number of vertices + number of edges). The x-axis shows all optimizations enabled. The y-axis of (g)–(k) shows one optimization disabled: (g) include premise of aug2, (h) include premise 1 of BFS-loop-it-init-c and disable immediate pruning of failed loop search iterations, (i) include premise 2 of BFS-loop-it-init-c, (j) include premise 2 of BFS-loop-conclusion2, (k) disable pruning of the resolution graph between loop searches. The y-axis of (l) shows all optimizations disabled.

IX. Conclusions

In this paper we showed how to obtain UCs for LTL via temporal resolution, and we demonstrated with an implementation in TRP++ that UC extraction can be performed efficiently. The resulting UCs are significantly smaller than the corresponding input formulas. In parallel work [Sch13] this paper has been used as a basis to suggest enhancing UCs for LTL with information on when subformulas of a UC are relevant for unsatisfiability. The similarity of temporal resolution and some BDD-based algorithms at a high level and work on resolution with BDDs ([JSB06]) suggests to explore whether computation of UCs is feasible for BDD-based algorithms. Another direction for transfer of our results is resolution-based computation of unrealizable cores [Noë95].

Acknowledgments

I thank B. Konev and M. Ludwig for making TRP++ and TSPASS including their LTL translators available. I also thank A. Cimatti for bringing up the subject of temporal resolution. Initial parts of the work were performed while working under a grant by the Provincia Autonoma di Trento (project EMTELOS).

References

[Arm+03] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. "Enhanced Vacuity Detection in Linear Temporal Logic". In: *CAV*. Ed. by W. Hunt Jr. and F. Somenzi. Vol. 2725. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2003, pp. 368–380. ISBN: 3-540-40524-0 (cit. on pp. 1, 2).

[Awa+12] A. Awad, R. Goré, Z. Hou, J. Thomson, and M. Weidlich. "An iterative approach to synthesize business process templates from compliance rules". In: *Inf. Syst.* 37.8 (2012). Links: ee, Google Scholar, pp. 714–736 (cit. on pp. 1, 2).

[Bak+93] R. Bakker, F. Dikker, F. Tempelman, and P. Wognum. "Diagnosing and Solving Over-Determined Constraint Satisfaction Problems". In: *IJCAI*. Links: ee, Google Scholar. 1993, pp. 276–281 (cit. on pp. 1, 6).

[BDF09] A. Behdenna, C. Dixon, and M. Fisher. "Deductive Verification of Simple Foraging Robotic Behaviours". In: *International Journal of Intelligent Computing and Cybernetics* 2.4 (2009). Links: ee, Google Scholar, pp. 604–643 (cit. on p. 8).

[Bee+01] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in Temporal Model Checking". In: *Formal Methods in System Design* 18.2 (2001). Links: ee, Google Scholar, pp. 141–163 (cit. on pp. 1, 2).

[BG01] L. Bachmair and H. Ganzinger. "Resolution Theorem Proving". In: *Handbook of Automated Reasoning*. Ed. by J. Robinson and A. Voronkov. Links: Google Scholar. Elsevier and MIT Press, 2001, pp. 19–99. ISBN: 0-444-50813-9, 0-262-18223-8 (cit. on pp. 1, 3, 7, 22).

[bgl] http://www.boost.org/doc/libs/release/libs/graph/ (cit. on p. 7).

[Bie+99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. "Symbolic Model Checking without BDDs". In: *TACAS*. Ed. by R. Cleaveland. Vol. 1579. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 1999, pp. 193–207. ISBN: 3-540-65703-7 (cit. on p. 2).

[Bie09] A. Biere. “Bounded Model Checking”. In: *Handbook of Satisfiability*. Ed. by A. Biere, M. Heule, H. van Maaren, and T. Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. Links: ee, Google Scholar. IOS Press, 2009, pp. 457–481. ISBN: 978-1-58603-929-5 (cit. on p. 2).

[Blo+07] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. “Specify, Compile, Run: Hardware from PSL”. In: *COCV*. Ed. by S. Glesner, J. Knoop, and R. Drechsler. Vol. 190(4). ENTCS. Links: ee, Google Scholar. Elsevier, 2007, pp. 3–16 (cit. on pp. 1, 8).

[BS01] R. Bruni and A. Sassano. “Restoring Satisfiability or Maintaining Unsatisfiability by finding small Unsatisfiable Subformulae”. In: *SAT*. Ed. by H. Kautz and B. Selman. Vol. 9. Electronic Notes in Discrete Mathematics. Links: ee, Google Scholar. Elsevier, 2001, pp. 162–173 (cit. on p. 1).

[CD91] J. Chinneck and E. Dravnieks. “Locating Minimal Infeasible Constraint Sets in Linear Programs”. In: *ORSA Journal on Computing* 3.2 (1991). Links: Google Scholar, pp. 157–168 (cit. on pp. 1, 6).

[Chi+10] A. Chiappini, A. Cimatti, L. Macchi, O. Rebollo, M. Roveri, A. Susi, S. Tonetta, and B. Vittorini. “Formalization and validation of a subset of the European Train Control System”. In: *ICSE (2)*. Ed. by J. Kramer, J. Bishop, P. Devanbu, and S. Uchitel. Links: ee, Google Scholar. ACM, 2010, pp. 109–118. ISBN: 978-1-60558-719-6 (cit. on p. 1).

[Cim+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. “NuSMV 2: An OpenSource Tool for Symbolic Model Checking”. In: *CAV*. Ed. by E. Brinksma and K. Larsen. Vol. 2404. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2002, pp. 359–364. ISBN: 3-540-43997-8 (cit. on p. 2).

[Cim+07] A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. “Boolean Abstraction for Temporal Logic Satisfiability”. In: *CAV*. Ed. by W. Damm and H. Hermanns. Vol. 4590. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2007, pp. 532–546. ISBN: 3-540-22342-8 (cit. on pp. 1, 2).

[Cim+08] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. “Diagnostic Information for Realizability”. In: *VMCAI*. Ed. by F. Logozzo, D. Peled, and L. Zuck. Vol. 4905. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2008, pp. 52–67. ISBN: 978-3-540-78162-2 (cit. on pp. 1, 2).

[Cla+03] E. Clarke, M. Talupur, H. Veith, and D. Wang. “SAT Based Predicate Abstraction for Hardware Verification”. In: *SAT*. Ed. by E. Giunchiglia and A. Tacchella. Vol. 2919. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2003, pp. 78–92. ISBN: 3-540-20851-8 (cit. on p. 1).

[CMT11] A. Cimatti, S. Mover, and S. Tonetta. “Proving and explaining the unfeasibility of message sequence charts for hybrid systems”. In: *FMCAD*. Ed. by P. Bjesse and A. Slobodová. Links: ee, Google Scholar. FMCAD Inc., 2011, pp. 54–62. ISBN: 978-0-9835678-1-3 (cit. on p. 2).

[Dix98] C. Dixon. “Temporal Resolution Using a Breadth-First Search Algorithm”. In: *Ann. Math. Artif. Intell.* 22.1-2 (1998). Links: ee, Google Scholar, pp. 87–115 (cit. on p. 15).

[DW+08] M. De Wulf, L. Doyen, N. Maquet, and J. Raskin. “Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking”. In: *TACAS*. Ed. by C. Ramakrishnan and J. Rehof. Vol. 4963. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2008, pp. 63–77. ISBN: 978-3-540-78799-0 (cit. on pp. 7, 8).

[Eas+03] S. Easterbrook, M. Chechik, B. Devereux, A. Gurfinkel, A. Lai, V. Petrovykh, A. Tafliovich, and C. Thompson-Walsh. “χChek: A Model Checker for Multi-Valued Reasoning”. In: *ICSE*. Ed. by L. Clarke, L. Dillon, and W. Tichy. Links: ee, Google Scholar. IEEE Computer Society, 2003, pp. 804–805 (cit. on p. 22).

[EF06] C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Links: Google Scholar. Springer, 2006 (cit. on p. 1).

[Eme90] E. Emerson. “Temporal and Modal Logic”. In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*. Ed. by J. van Leeuwen. Links: Google Scholar. Elsevier and MIT Press, 1990, pp. 995–1072. ISBN: 0-444-88074-7, 0-262-22039-3 (cit. on pp. 1, 2).

[FDP01] M. Fisher, C. Dixon, and M. Peim. “Clausal temporal resolution”. In: *ACM Trans. Comput. Log.* 2.1 (2001). Links: ee, Google Scholar, pp. 12–56 (cit. on pp. 2, 3, 5, 15).

[Fis+08] D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M. Vardi. “A Framework for Inherent Vacuity”. In: *HVC*. Ed. by H. Chockler and A. Hu. Vol. 5394. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2008, pp. 7–22. ISBN: 978-3-642-01701-8 (cit. on pp. 1, 2).

[Fis91] M. Fisher. “A Resolution Method for Temporal Logic”. In: *IJCAI*. Links: ee, Google Scholar. 1991, pp. 99–104 (cit. on pp. 2, 3).

[FM09] J. Franco and J. Martin. “A History of Satisfiability”. In: *Handbook of Satisfiability*. Ed. by A. Biere, M. Heule, H. van Maaren, and T. Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. Links: ee, Google Scholar. IOS Press, 2009, pp. 3–74. ISBN: 978-1-58603-929-5 (cit. on p. 3).

[FN92] M. Fisher and P. Noël. *Transformation and Synthesis in* METATEM. *Part I: Propositional* METATEM. Tech. rep. UMCS-92-2-1. Available from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.4998. University of Manchester, Department of Computer Science, 1992 (cit. on p. 3).

[GC04] A. Gurfinkel and M. Chechik. “How Vacuous Is Vacuous?” In: *TACAS*. Ed. by K. Jensen and A. Podelski. Vol. 2988. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2004, pp. 451–466. ISBN: 3-540-21299-X (cit. on pp. 1, 2, 21, 22).

[GG06] M. Gheorghiu and A. Gurfinkel. “VaqUoT: A Tool for Vacuity Detection”. In: *FM*. Ed. by J. Misra, T. Nipkow, and E. Sekerinski. Vol. 4085. Lecture Notes in Computer Science. Tool Presentation. Available from http://fm06.mcmaster.ca/VaqUoT.pdf. Springer, 2006. ISBN: 3-540-37215-6 (cit. on p. 22).

[GN03] E. Goldberg and Y. Novikov. “Verification of Proofs of Unsatisfiability for CNF Formulas”. In: *DATE*. Links: ee, Google Scholar. IEEE Computer Society, 2003, pp. 10886–10891. ISBN: 0-7695-1870-2 (cit. on p. 1).

[Har05] A. Harding. “Symbolic Strategy Synthesis For Games With LTL Winning Conditions”. Links: Google Scholar. PhD thesis. University of Birmingham, 2005 (cit. on pp. 6, 8).

[Heu+95] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. “Propositional Logics on the Computer”. In: *TABLEAUX*. Ed. by P. Baumgartner, R. Hähnle, and J. Posegga. Vol. 918. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 1995, pp. 310–323. ISBN: 3-540-59338-1 (cit. on p. 2).

[HH11] F. Hantry and M. Hacid. “Handling Conflicts in Depth-First Search for LTL Tableau to Debug Compliance Based Languages”. In: *FLACOS*. Ed. by E. Pimentel and V. Valero. Vol. 68. Electronic Proceedings in Theoretical Computer Science. Links: ee, Google Scholar. Open Publishing Association, 2011, pp. 39–53 (cit. on pp. 1, 2).

[HK03] U. Hustadt and B. Konev. “TRP++ 2.0: A Temporal Resolution Prover”. In: *CADE*. Ed. by F. Baader. Vol. 2741. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2003, pp. 274–278. ISBN: 3-540-40559-3 (cit. on pp. 2, 3, 7, 22).

[HK04] U. Hustadt and B. Konev. “TRP++: A temporal resolution prover”. In: *Collegium Logicum*. Ed. by M. Baaz, J. Makowsky, and A. Voronkov. Vol. 8. Links: Google Scholar. Kurt Gödel Society, 2004, pp. 65–79 (cit. on pp. 2, 3).

[Hoo99] H. Hoos. *Heavy-Tailed Behaviour in Randomised Systematic Search Algorithms for SAT?* Tech. rep. TR-99-16. Links: ee, Google Scholar. University of British Columbia, Department of Computer Science, 1999 (cit. on p. 1).

[HS02] U. Hustadt and R. A. Schmidt. “Scientific Benchmarking with Temporal Logic Decision Procedures”. In: *KR*. Ed. by D. Fensel, F. Giunchiglia, D. McGuinness, and M. Williams. Links: Google Scholar. Morgan Kaufmann, 2002, pp. 533–546. ISBN: 1-55860-554-1 (cit. on p. 8).

[HSH12] F. Hantry, L. Saïs, and M. Hacid. “On the complexity of computing minimal unsatisfiable LTL formulas”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 19.69 (2012). Links: ee, Google Scholar (cit. on pp. 1, 2).

[JB06] B. Jobstmann and R. Bloem. “Optimizations for LTL Synthesis”. In: *FMCAD*. Links: ee, Google Scholar. IEEE Computer Society, 2006, pp. 117–124. ISBN: 0-7695-2707-8 (cit. on p. 6).

[Jos12] N. Josuttis. *The C++ Standard Library: A Tutorial and Reference*. Second Edition. Links: Google Scholar. Addison-Wesley, 2012. ISBN: 978-0-321-62321-8 (cit. on p. 7).

[JSB06] T. Jussila, C. Sinz, and A. Biere. “Extended Resolution Proofs for Symbolic SAT Solving with Quantification”. In: *SAT*. Ed. by A. Biere and C. Gomes. Vol. 4121. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2006, pp. 54–60. ISBN: 3-540-37206-7 (cit. on p. 9).

[Jun01] U. Junker. “QuickXplain: Conflict Detection for Arbitrary Constraint Propagation Algorithms”. In: *CONS*. Available from http://www.lirmm.fr/~bessiere/ws_ijcai01/junker.ps.gz. 2001 (cit. on p. 6).

[KHB09] R. Könighofer, G. Hofferek, and R. Bloem. “Debugging formal specifications using simple counterstrategies”. In: *FMCAD*. Links: ee, Google Scholar. IEEE, 2009, pp. 152–159. ISBN: 978-1-4244-4966-8 (cit. on pp. 2, 6).

[KHB10] R. Könighofer, G. Hofferek, and R. Bloem. “Debugging Unrealizable Specifications with Model-Based Diagnosis”. In: *HVC*. Ed. by S. Barner, I. Harris, D. Kroening, and O. Raz. Vol. 6504. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2010, pp. 29–45. ISBN: 978-3-642-19582-2 (cit. on p. 2).

[Kup06] O. Kupferman. “Sanity Checks in Formal Verification”. In: *CONCUR*. Ed. by C. Baier and H. Hermanns. Vol. 4137. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2006, pp. 37–51. ISBN: 3-540-37376-4 (cit. on pp. 1, 2).

[KV03] O. Kupferman and M. Vardi. “Vacuity detection in temporal model checking”. In: *STTT* 4.2 (2003). Links: ee, Google Scholar, pp. 224–233 (cit. on pp. 1, 2).

[LH10] M. Ludwig and U. Hustadt. “Implementing a fair monodic temporal logic prover”. In: *AI Commun.* 23.2-3 (2010). Links: ee, Google Scholar, pp. 69–96 (cit. on p. 7).

[MS10] J. Marques Silva. “Minimal Unsatisfiability: Models, Algorithms and Applications (Invited Paper)”. In: *ISMVL*. Links: ee, Google Scholar. IEEE Computer Society, 2010, pp. 9–14. ISBN: 978-0-7695-4024-5 (cit. on pp. 1, 6).

[Nam04] K. Namjoshi. “An Efficiently Checkable, Proof-Based Formulation of Vacuity in Model Checking”. In: *CAV*. Ed. by R. Alur and D. Peled. Vol. 3114. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2004, pp. 57–69. ISBN: 3-540-22342-8 (cit. on p. 22).

[nma] http://code.google.com/a/eclipselabs.org/p/nusmv-tools/downloads/detail?name=NuSMVModelAdvisor_20121012.zip (cit. on p. 22).

[Noë95] P. Noël. “A Transformation-Based Synthesis of Temporal Specifications”. In: *Formal Asp. Comput.* 7.6 (1995). Links: ee, Google Scholar, pp. 587–619 (cit. on p. 9).

[PA06] M. Pesic and W. van der Aalst. “A Declarative Approach for Flexible Business Processes Management”. In: *Business Process Management Workshops*. Ed. by J. Eder and S. Dustdar. Vol. 4103. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2006, pp. 169–180. ISBN: 3-540-38444-8 (cit. on p. 1).

[PG86] D. Plaisted and S. Greenbaum. “A Structure-Preserving Clause Form Translation”. In: *J. Symb. Comput.* 2.3 (1986). Links: ee, Google Scholar, pp. 293–304 (cit. on p. 5).

[Pil+06] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. “Formal analysis of hardware requirements”. In: *DAC*. Ed. by E. Sentovich. Links: ee, Google Scholar. ACM, 2006, pp. 821–826. ISBN: 1-59593-381-6 (cit. on p. 1).

[pltl] http://users.cecs.anu.edu.au/~rpg/PLTLProvers/ (cit. on p. 2).

[Pnu77] A. Pnueli. “The Temporal Logic of Programs”. In: *FOCS*. Links: ee, Google Scholar. IEEE, 1977, pp. 46–57 (cit. on p. 1).

[PWK09] M. Purandare, T. Wahl, and D. Kroening. “Strengthening properties using abstraction refinement”. In: *DATE*. Links: ee, Google Scholar. IEEE, 2009, pp. 1692–1697 (cit. on p. 21).

[RKG11] V. Raman and H. Kress-Gazit. “Analyzing Unsynthesizable Specifications for High-Level Robot Behavior Using LTLMoP”. In: *CAV*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2011, pp. 663–668. ISBN: 978-3-642-22109-5 (cit. on p. 2).

[Rob65] J. Robinson. “A Machine-Oriented Logic Based on the Resolution Principle”. In: *J. ACM* 12.1 (1965). Links: ee, Google Scholar, pp. 23–41 (cit. on pp. 1, 3).

[run] A. Biere and T. Jussila. *Benchmark Tool Run*. http://fmv.jku.at/run/ (cit. on p. 8).

[RV10] K. Rozier and M. Vardi. “LTL satisfiability checking”. In: *STTT* 12.2 (2010). Links: ee, Google Scholar, pp. 123–137 (cit. on p. 8).

[Sch12] V. Schuppan. “Towards a notion of unsatisfiable and unrealizable cores for LTL”. In: *Sci. Comput. Program.* 77.7-8 (2012). Links: ee, Google Scholar, pp. 908–939 (cit. on pp. 1, 2, 6, 21).

[Sch13] V. Schuppan. “Enhancing Unsatisfiable Cores for LTL with Information on Temporal Relevance”. In: *QAPL*. Ed. by L. Bortolussi and H. Wiklicky. Vol. 117. Electronic Proceedings in Theoretical Computer Science. Links: ee, Google Scholar. Open Publishing Association, 2013, pp. 49–65 (cit. on pp. 2, 9).

[SD11] V. Schuppan and L. Darmawan. “Evaluating LTL Satisfiability Solvers”. In: *ATVA*. Ed. by T. Bultan and P. Hsiung. Vol. 6996. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 2011, pp. 397–413. ISBN: 978-3-642-24371-4 (cit. on pp. 2, 7, 8).

[SDG06] J. Simmonds, J. Davies, and A. Gurfinkel. “VaqTree: Efficient Vacuity Detection for Bounded Model Checking”. In: *FM*. Ed. by J. Misra, T. Nipkow, and E. Sekerinski. Vol. 4085. Lecture Notes in Computer Science. Tool Presentation. Available from http://fm06.mcmaster.ca/jocelyn.pdf. Springer, 2006. ISBN: 3-540-37215-6 (cit. on p. 21).

[Shl+03] I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. “Debugging Overconstrained Declarative Models Using Unsatisfiable Cores”. In: *ASE*. Links: ee, Google Scholar. IEEE Computer Society, 2003, pp. 94–105. ISBN: 0-7695-2035-9 (cit. on p. 1).

[Sim+10] J. Simmonds, J. Davies, A. Gurfinkel, and M. Chechik. “Exploiting resolution proofs to speed up LTL vacuity detection for BMC”. In: *STTT* 12.5 (2010). Links: ee, Google Scholar, pp. 319–335 (cit. on pp. 1, 2, 21).

[SL95] A. Stepanov and M. Lee. *The Standard Template Library*. Tech. rep. 95-11(R.1). Links: Google Scholar. HP Laboratories, Nov. 1995. URL: http://www.stepanovpapers.com/STL/DOC.PDF (cit. on p. 7).

[SLL02] J. Siek, L. Lee, and A. Lumsdaine. *The Boost Graph Library - User Guide and Reference Manual*. C++ in-depth series. Links: Google Scholar. Pearson / Prentice Hall, 2002, pp. I–XXIV, 1–321. ISBN: 978-0-201-72914-6 (cit. on p. 7).

[trp++] http://www.csc.liv.ac.uk/~konev/software/trp++/ (cit. on pp. 2, 3).

[VG02] A. Van Gelder. “Extracting (Easily) Checkable Proofs from a Satisfiability Solver that Employs both Preorder and Postorder Resolution”. In: *AMAI*. Links: ee, Google Scholar. 2002 (cit. on p. 1).

[vis96] The VIS Group. “VIS: A System for Verification and Synthesis”. In: *CAV*. Ed. by R. Alur and T. Henzinger. Vol. 1102. Lecture Notes in Computer Science. Links: ee, Google Scholar. Springer, 1996, pp. 428–432. ISBN: 3-540-61474-5 (cit. on p. 21).

[Wha94] D. Whalley. “Automatic Isolation of Compiler Errors”. In: *ACM Trans. Program. Lang. Syst.* 16.5 (1994). Links: ee, Google Scholar, pp. 1648–1659 (cit. on p. 1).

[www] http://www.schuppan.de/viktor/time13/ (cit. on pp. 2, 7).

[ZH02] A. Zeller and R. Hildebrandt. “Simplifying and Isolating Failure-Inducing Input”. In: *IEEE Trans. Software Eng.* 28.2 (2002). Links: ee, Google Scholar, pp. 183–200 (cit. on pp. 1, 6).

[ZM03a] L. Zhang and S. Malik. *Extracting Small Unsatisfiable Cores from Unsatisfiable Boolean Formula*. Presented at *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003*. Links: Google Scholar. 2003 (cit. on p. 6).

[ZM03b] L. Zhang and S. Malik. “Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications”. In: *DATE*. Links: ee, Google Scholar. IEEE Computer Society, 2003, pp. 10880–10885. ISBN: 0-7695-1870-2 (cit. on p. 1).

APPENDIX A
PROOFS: IV UC EXTRACTION

Lemma 1. *Let C be a set of SNF clauses to which the algorithm in Fig. 2 has been applied and shown unsatisfiability, let G be the resolution graph, and let G' the subgraph according to Def. 2. Let v be a vertex in G' labeled with a clause $c = (\mathbf{G}((\neg wl) \vee (\mathbf{X}(l \vee wl))))$ created by augmentation* aug2 *from some eventuality clause $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{F}(l)))) \in C$ with eventuality literal l. Then there is a vertex v' in G' labeled with an eventuality clause $c' = (\mathbf{G}((q_1 \vee \ldots \vee q_{n'}) \vee (\mathbf{F}(l)))) \in C$ with eventuality literal l.*

Proof: There exists a path π of non-zero length in G' from v to the unique vertex $v_\square$ in the main partition M labeled with the empty clause $\square$. On the path π there exist two vertices v'', v''' such that v'' is labeled with a clause c'' that contains $\neg wl$ or $\mathbf{X}\neg wl$, while v''' and all of its successors on π are labeled with clauses that contain neither $\neg wl$ nor $\mathbf{X}\neg wl$. Let c''' be the clause labeling v'''.

- *Case 1.* c''' is generated by initial or step resolution init-ii, init-in, step-nn, step-nx, or step-xx from c'' and some other clause c''''. c'''' must contain wl or $\mathbf{X}wl$. Moreover, there must be a path π' (possibly of zero length) that starts from a vertex v''''' labeled with a clause c''''' and that ends in the vertex v'''' labeled with c'''', such that each vertex on the path π' is labeled with a clause that contains wl or $\mathbf{X}wl$. Finally, wl or $\mathbf{X}wl$ must be present in c''''' either because c''''' is contained in the set of input clauses in SNF, C, or because c''''' is generated by some production rule that introduces wl or $\mathbf{X}wl$ in the conclusion.
  - *Case 1.1.* c''''' is contained in the set of input clauses in SNF, C. Impossible: wl is a fresh proposition in aug1 and aug2.
  - *Case 1.2.* c''''' is generated by initial or step resolution init-ii, init-in, step-nn, step-nx, or step-xx. Impossible: initial and step resolution do not generate literals that are not contained (modulo time-shifting) in at least one of the premises.
  - *Case 1.3.* c''''' is generated by augmentation 1 aug1. By construction of the resolution graph G and the subgraph G' there is an edge in G' from a vertex v' in G' labeled with an eventuality clause $c' = (\mathbf{G}((q_1 \vee \ldots \vee q_{n'}) \vee (\mathbf{F}(l)))) \in C$ with eventuality literal l to v'''''.
  - *Case 1.4.* c''''' is generated by augmentation 2 aug2, i.e., $c''''' = c$. This introduces another occurrence of $\neg wl$ to be "resolved away". Note that in the main partition only new clauses are generated from existing ones with edges leading from existing vertices labeled with existing clauses to new vertices labeled with new clauses. Therefore, the main partition of G' is a finite directed acyclic graph, and this case cannot happen infinitely often.
  - *Case 1.5.* c''''' is generated by BFS loop search initialization BFS-loop-it-init-x. Impossible: the production rule BFS-loop-it-init-x copies a clause verbatim. I.e., it cannot be the case that c''''' contains wl or $\mathbf{X}wl$, while the premise does not.
  - *Case 1.6.* c''''' is generated by BFS loop search initialization BFS-loop-it-init-n. Impossible: the production rule BFS-loop-it-init-n copies and time-shifts a clause. I.e., it cannot be the case that c''''' contains $\mathbf{X}wl$, while the premise does not contain wl.
  - *Case 1.7.* c''''' is generated by BFS loop search initialization BFS-loop-it-init-c. Impossible: the production rule BFS-loop-it-init-c copies and time-shifts a clause from a previous BFS loop search iteration (or initializes with the empty clause $\square$) and disjoins with an eventuality literal $\mathbf{X}l'$. I.e., it cannot be the case that c''''' contains $\mathbf{X}wl$, while the premise does not contain wl.
  - *Case 1.8.* v''''' is linked to via BFS loop search subsumption BFS-loop-it-sub. This case can be ignored as BFS loop search subsumption BFS-loop-it-sub does not actually generate a clause but merely links existing ones.
  - *Case 1.9.* c''''' is generated by BFS loop search conclusion 1 BFS-loop-conclusion1. Impossible: production rule BFS-loop-conclusion1 copies all literals verbatim from a clause derived in loop search, copies all literals verbatim from an eventuality clause except for the eventuality literal l' prefixed by $\mathbf{F}$, and disjoins with the eventuality literal l'. I.e., it cannot be the case that c''''' contains wl, while the premises do not.
  - *Case 1.10.* c''''' is generated by BFS loop search conclusion 2 BFS-loop-conclusion2. Impossible: production rule BFS-loop-conclusion2 copies and time-shifts all literals from a clause c'''''' derived in loop search and disjoins with $\neg wl'$ and $\mathbf{X}l'$ for some eventuality literal l'. I.e., it cannot be the case that c''''' contains $\mathbf{X}wl$, while the premise c'''''' does not contain wl.
- *Case 2.* c''' is generated by augmentation aug1 or aug2. Impossible: the premise of the production rules aug1 and aug2 cannot contain either $\neg wl$ or $\mathbf{X}\neg wl$ as wl is assumed to be a fresh proposition in aug1 and aug2.
- *Case 3.* c''' is generated by BFS loop search initialization BFS-loop-it-init-x. Impossible: the production rule BFS-loop-it-init-x copies a clause verbatim. I.e., it cannot be the case that c'' contains $\neg wl$ or $\mathbf{X}\neg wl$, while c''' does not.
- *Case 4.* c''' is generated by BFS loop search initialization BFS-loop-it-init-n. Impossible: the production rule BFS-loop-it-init-n copies and time-shifts a clause. I.e., it cannot be the case that c'' contains $\neg wl$, while c''' does not contain $\mathbf{X}\neg wl$.
- *Case 5.* c''' is generated by BFS loop search initialization BFS-loop-it-init-c. Impossible: the production rule BFS-loop-it-init-c copies and time-shifts a clause from a previous BFS loop search iteration (or initializes with the empty clause $\square$) and disjoins with an eventuality literal $\mathbf{X}l'$. I.e., it cannot be the case that c'' contains $\neg wl$, while

c''' does not contain $\mathbf{X}\neg wl$.

- *Case 6.* v'' and v''' are linked via BFS loop search subsumption BFS-loop-it-sub, i.e., a time-shifted version of c'' subsumes c'''. Impossible: BFS-loop-it-sub links from a clause with fewer literals to a clause with (modulo time-shifting) the same and more literals. I.e., it cannot be the case that c'' contains $\neg wl$, while c''' does not contain $\mathbf{X}\neg wl$.
- *Case 7.* c''' is generated by BFS loop search conclusion 1 BFS-loop-conclusion1. Impossible: production rule BFS-loop-conclusion1 copies all literals verbatim from a clause derived in loop search, copies all literals verbatim from an eventuality clause except for the eventuality literal l' prefixed by $\mathbf{F}$, and disjoins with the eventuality literal l'. I.e., it cannot be the case that c'' contains $\neg wl$, while c''' does not.
- *Case 8.* c''' is generated by BFS loop search conclusion 2 BFS-loop-conclusion2. Impossible: production rule BFS-loop-conclusion2 copies and time-shifts all literals from a clause derived in loop search and disjoins with $\neg wl'$ and $\mathbf{X}l'$ for some eventuality literal l'. I.e., it cannot be the case that c'' contains $\neg wl$, while c''' does not contain $\mathbf{X}\neg wl$.

Notice that the only possible cases are case 1.3 and 1.4. Of those, case 1.4 can only happen a finite number of times and must be followed by an occurrence of case 1.3. This concludes the proof. ■

Lemma 2. *Let C be a set of SNF clauses to which the algorithm in Fig. 2 has been applied and shown unsatisfiability, let G be the resolution graph constructed, and let G' be the subgraph according to Def. 2. Let v be a vertex in G' labeled with a clause $c = (\mathbf{G}((\neg wl) \vee (\mathbf{X}((q_1 \vee \ldots \vee q_{n'}) \vee l))))$ generated by BFS loop search conclusion 2 BFS-loop-conclusion2 from some eventuality clause $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{F}(l)))) \in C$ with eventuality literal l (and some other clause). Then there is a vertex v'' in G' labeled with an eventuality clause $c'' = (\mathbf{G}((r_1 \vee \ldots \vee r_{n''}) \vee (\mathbf{F}(l)))) \in C$ with eventuality literal l.*

Proof: Analogous to the proof of Lemma 1. ■

Lemma 3. *Let C be a set of SNF clauses to which the algorithm in Fig. 2 has been applied and shown unsatisfiability, let G be the resolution graph, and let G' be the subgraph according to Def. 2. Let v be a vertex in G' labeled with a clause $c = (\mathbf{G}((0) \vee (\mathbf{X}(q_1 \vee \ldots \vee q_{n'} \vee l))))$ generated by production rule BFS-loop-it-init-c from some eventuality clause $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{F}(l)))) \in C$ with eventuality literal l (and some other clause). Then there is a vertex v'' in G' labeled with an eventuality clause $c'' = (\mathbf{G}((r_1 \vee \ldots \vee r_{n''}) \vee (\mathbf{F}(l)))) \in C$ with eventuality literal l.*

Proof: By construction of the resolution graph G (Def. 1) and its subgraph G' (Def. 2) v is included in G' only if G' also includes some vertex v' labeled with some clause c' such that c' was generated by BFS loop search conclusion BFS-loop-conclusion1 or BFS-loop-conclusion2 from the BFS loop search iteration of which c is part.

- *Case 1.* c' is generated by BFS loop search conclusion 1 BFS-loop-conclusion1. The claim follows from the construction of the resolution graph G and its subgraph G'. By Def. 1 v' has an incoming edge from a vertex v'' labeled with an eventuality clause $c'' = (\mathbf{G}((r_1 \vee \ldots \vee r_{n''}) \vee (\mathbf{F}(l)))) \in C$ with eventuality literal l and by Def. 2 v'' is included in G' if v' is included.
- *Case 2.* c' is generated by BFS loop search conclusion 2 BFS-loop-conclusion2. In that case the claim follows directly from Lemma 2.

This concludes the proof. ■

Theorem 1 (Unsatisfiability of UC in SNF)**.** *Let C be a set of SNF clauses to which the algorithm in Fig. 2 has been applied and shown unsatisfiability, and let C^{uc} be the UC of C in SNF. Then C^{uc} is unsatisfiable.*

Proof: Assume for a moment that in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I all ✖ are replaced with ✔, i.e., that each conclusion in the resolution graph is connected by an edge to each of its premises rather than only to a subset of them. In that case the UC in SNF according to Def. 2 would contain all clauses of the set of starting clauses C that contributed to deriving the empty clause and, hence, to establishing unsatisfiability of C. In that case it would follow directly from the correctness of TR that C^{uc} is unsatisfiable.

It remains to show that (i) not including an edge from premise 1 to the conclusion for rule aug2, (ii) not including an edge from premise 2 to the conclusion for rule BFS-loop-conclusion2, (iii) not including an edge from premise 2 to the conclusion for rule BFS-loop-it-init-c, and (iv) not including an edge from premise 1 to the conclusion for rule BFS-loop-it-init-c in the resolution graph G maintains the fact that the resulting C^{uc} is unsatisfiable.

To see the intuition behind (i) note that for a vertex v_c labeled with a conclusion c of rule aug2 in the main partition M^V to be backward reachable from the (unique) vertex in the main partition M^V of the resolution graph G labeled with the empty clause $\square$, $v_\square$, the occurrence of $\neg wl$ in c must be "resolved away" at some point on the path from v_c to $v_\square$. It turns out that this can only happen by resolution with a clause that is derived from the conclusion of rule aug1 applied to an eventuality clause c' with eventuality literal l. By construction of the resolution graph G $v_{c'}$ must be backward reachable from $v_\square$ and, therefore, c' must be included in the UC in SNF. Hence, an execution of the algorithm in Fig. 2 with input C^{uc} will produce c from c'. For a formal proof see Lemma 1.

A similar reasoning as for (i) applies to (ii), formalized in Lemma 2.

For (iii) note that a conclusion of rule BFS-loop-it-init-c can only be backward reachable from $v_\square$ if the corresponding BFS loop search iteration is successful and a vertex labeled with one of the resulting conclusions of rules BFS-loop-conclusion1 or BFS-loop-conclusion2 is backward reachable from $v_\square$. The latter

fact implies that an eventuality clause with the same eventuality literal as in premise 2 of rule BFS-loop-it-init-c is present in the UC in SNF. Hence, an execution of the algorithm in Fig. 2 with input C^{uc} will produce premise 2 of BFS-loop-it-init-c as required. This is formally proven in Lemma 3.

Finally, (iv) is obtained by understanding that in a BFS loop search iteration the premises 1 of rule BFS-loop-it-init-c essentially constitute a hypothetical fixed point; if the BFS loop search iteration is successful, then the hypothetical fixed point is proven to be an actual fixed point. For the correctness of a proof of unsatisfiability of C it is only relevant that this hypothetical fixed point is shown to be an actual fixed point but not how the hypothesis is obtained. This is formalized below.

Notice that (iv) essentially corresponds to considering only the last iteration of a successful loop search to obtain the UC C^{uc}. After initialization of a loop search iteration in line 11 of the algorithm in Fig. 2 L contains three sets of clauses according to the three production rules for initializing a loop search iteration. Clauses generated by BFS-loop-it-init-x and BFS-loop-it-init-n are (partly time-shifted) duplicates of clauses derived so far in the main partition. BFS-loop-it-init-c generates a set of clauses $(\mathbf{G}((0) \vee (\mathbf{X}(p_{i,1} \vee \ldots \vee p_{i,n_i} \vee l))))$. From these three sets saturation restricted to rule step-xx in line 12 derives another set of clauses $(\mathbf{G}(q_{i',1} \vee \ldots \vee q_{i',n'_{i'}}))$. Taking the restriction of saturation to rule step-xx into account, that loop search iteration has established that, assuming C, the following fact is provable:

$$\mathbf{G}((\bigwedge_{1\leq i\leq n}(\mathbf{X}(p_{i,1} \vee \ldots \vee p_{i,n_i} \vee l))) \rightarrow (\bigwedge_{1\leq i'\leq n'}(q_{i',1} \vee \ldots \vee q_{i',n'_{i'}}))) \tag{10}$$

Moreover, if subsumption in line 15 succeeds, the following fact is also provable:

$$\bigwedge_{1\leq i\leq n}(\bigvee_{1\leq i'\leq n'}(\mathbf{G}((q_{i',1} \vee \ldots \vee q_{i',n'_{i'}}) \rightarrow (p_{i,1} \vee \ldots \vee p_{i,n_i})))) \tag{11}$$

We rewrite (10) and (11) as follows:

$$\begin{aligned}
&\mathbf{G}((\textstyle\bigwedge_{1\leq i\leq n}(\mathbf{X}(p_{i,1} \vee \ldots \vee p_{i,n_i} \vee l))) \rightarrow (\bigwedge_{1\leq i'\leq n'}(q_{i',1} \vee \ldots \vee q_{i',n'_{i'}}))) \\
\Leftrightarrow\ &\mathbf{G}(\textstyle\bigwedge_{1\leq i'\leq n'}((\bigwedge_{1\leq i\leq n}(\mathbf{X}(p_{i,1} \vee \ldots \vee p_{i,n_i} \vee l))) \rightarrow (q_{i',1} \vee \ldots \vee q_{i',n'_{i'}}))) \\
\Leftrightarrow\ &\textstyle\bigwedge_{1\leq i'\leq n'}(\mathbf{G}((\bigwedge_{1\leq i\leq n}(\mathbf{X}(p_{i,1} \vee \ldots \vee p_{i,n_i} \vee l))) \rightarrow (q_{i',1} \vee \ldots \vee q_{i',n'_{i'}}))) \\
\Leftrightarrow\ &\textstyle\bigwedge_{1\leq i'\leq n'}(\mathbf{G}((\neg(q_{i',1} \vee \ldots \vee q_{i',n'_{i'}})) \rightarrow (\neg(\bigwedge_{1\leq i\leq n}(\mathbf{X}(p_{i,1} \vee \ldots \vee p_{i,n_i} \vee l)))))) \\
\Leftrightarrow\ &\textstyle\bigwedge_{1\leq i'\leq n'}(\mathbf{G}((\neg(q_{i',1} \vee \ldots \vee q_{i',n'_{i'}})) \rightarrow (\bigvee_{1\leq i\leq n}(\mathbf{X}(\neg(p_{i,1} \vee \ldots \vee p_{i,n_i} \vee l)))))) \\
\Leftrightarrow\ &\textstyle\bigwedge_{1\leq i'\leq n'}(\mathbf{G}((\neg(q_{i',1} \vee \ldots \vee q_{i',n'_{i'}})) \rightarrow (\bigvee_{1\leq i\leq n}(\mathbf{X}((\neg(p_{i,1} \vee \ldots \vee p_{i,n_i})) \wedge (\neg l)))))) \\
\Leftrightarrow\ &\textstyle\bigwedge_{1\leq i'\leq n'}(\mathbf{G}((\neg(q_{i',1} \vee \ldots \vee q_{i',n'_{i'}})) \rightarrow ((\mathbf{X}\neg l) \wedge (\bigvee_{1\leq i\leq n}(\mathbf{X}(\neg(p_{i,1} \vee \ldots \vee p_{i,n_i}))))))
\end{aligned} \tag{12}$$

$$\begin{aligned}
&\textstyle\bigwedge_{1\leq i\leq n}(\bigvee_{1\leq i'\leq n'}(\mathbf{G}((q_{i',1} \vee \ldots \vee q_{i',n'_{i'}}) \rightarrow (p_{i,1} \vee \ldots \vee p_{i,n_i})))) \\
\Leftrightarrow\ &\textstyle\bigwedge_{1\leq i\leq n}(\bigvee_{1\leq i'\leq n'}(\mathbf{G}((\neg(p_{i,1} \vee \ldots \vee p_{i,n_i})) \rightarrow (\neg(q_{i',1} \vee \ldots \vee q_{i',n'_{i'}}))))))
\end{aligned} \tag{13}$$

Putting (12) and (13) together, we obtain (14), which is exactly the premise required to perform eventuality resolution with an eventuality clause with eventuality literal l [FDP01]:

$$\begin{aligned}
&(\mathbf{G}((q_{1,1} \vee \ldots \vee q_{1,n'_1}) \vee (\mathbf{X}\mathbf{G}\neg l))) \\
&\ldots \\
&(\mathbf{G}((q_{n',1} \vee \ldots \vee q_{n',n'_{n'}}) \vee (\mathbf{X}\mathbf{G}\neg l)))
\end{aligned} \tag{14}$$

This concludes the proof. ■

Proposition 1 (Complexity of UC Extraction)**.** *Let C be a set of SNF clauses to which the algorithm in Fig. 2 is applied and shows unsatisfiability. Construction and backward traversal of the resolution graph and, hence, construction of C^{uc} according to Def. 2 can be performed in time $\mathcal{O}(|V|)$ in addition to the time required to run the algorithm in Fig. 2. $|V|$ is at most exponential in $|AP| + log(|C|)$.*

Proof: Notice that each vertex in G has at most 2 incoming edges. Hence, construction of G and backward traversal of G from the unique vertex in the main partition labeled with the empty clause, $v_\square$, can be performed in time $\mathcal{O}(|V|)$.

For a proof of $|AP|+log(|C|)$ see the following reasoning:

1) In an initial clause a proposition can be not present, present, or present negated. Hence, the number of different initial clauses is $\mathcal{O}(3^{|AP|})$.
2) In a global clause a proposition can be one of not present, present, or present negated; and prefixed by $\mathbf{X}$ not present, present, or present negated. Hence, the number of different global clauses is $\mathcal{O}(9^{|AP|})$.
3) The number of clauses in the main partition is bounded by $|C| + \mathcal{O}(3^{|AP|}) + \mathcal{O}(9^{|AP|}) = \mathcal{O}(|C| + 9^{|AP|})$.
4) The number of clauses in a partition for a BFS loop search iteration is bounded by $\mathcal{O}(9^{|AP|})$.
5) The number of partitions is bounded by 1 plus the number of BFS loop search iterations.
6) The number of iterations in a BFS loop search is bounded by the length of the longest monotonically increasing sequence of Boolean formulas over AP, which is $\mathcal{O}(2^{|AP|})$. See also [Dix98].
7) The number of BFS loop searches is bounded by the number of different clauses that can be the result of a BFS loop search. The number of different clauses that can be the consequence of BFS loop search conclusion 1 BFS-loop-conclusion1 is bounded by the number of different global clauses with empty next part, which is $\mathcal{O}(3^{|AP|})$. The number of different clauses that can be the consequence of BFS loop search conclusion 2 BFS-loop-conclusion2 is bounded by the number of different eventuality literals times the number of different global clauses with empty

next part, which is $\mathcal{O}(|C| \cdot 3^{|AP|})$. Hence, the number of BFS loop searches is bounded by $\mathcal{O}(|C| \cdot 3^{|AP|})$.

8) Taking all of the above into account, the number of clauses is bounded by $\mathcal{O}(|C| + 9^{|AP|} + |C| \cdot 3^{|AP|} \cdot 2^{|AP|} \cdot 9^{|AP|}) = \mathcal{O}(|C| \cdot 54^{|AP|})$.

This concludes the proof. ■

Remark 1 (Minimality of Set of Premises to Include in Resolution Graph)**.** *Theorem 1 shows that not including premises for production rules marked by ✖ in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I during the construction of the resolution graph still leads to a UC. It does not discuss whether the remaining premises, marked by ✔ in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I, actually need to be included to guarantee a UC. For all premises of all production rules marked by ✔ in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I it turns out that they are indeed required to obtain a UC.*

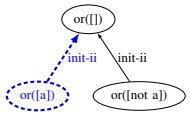
Proof: For a given premise p of some production rule [rule] with conclusion c the need for inclusion of edges between instances of p and c induced by [rule] in the resolution graph to obtain a UC can be established as follows.[2] Let C be a set of SNF clauses to which the algorithm in Fig. 2 has been applied and shown unsatisfiability, let G be the resolution graph, let $v_\square$ be the (unique) vertex in the main partition M^V of the resolution graph G labeled with the empty clause $\square$, and let C^{uc} be the UC of C in SNF. Assume that C^{uc} is a minimal UC. Now, if removing all edges between instances of p and c induced by [rule] from G makes a vertex v_c in M^V labeled with $c \in C^{uc}$ backward unreachable from $v_\square$, then the UC obtained without including edges between instances of p and c induced by [rule] clearly would not be unsatisfiable.

Hence, for all premises of all production rules marked by ✔ in columns 9 (*p.1 – c*) and 10 (*p.2 – c*) of Tab. I, below we provide triples of premises p of production rules [rule], minimally unsatisfiable SNFs $C \equiv C^{uc}$, and subgraphs G' of resolution graphs with the following properties. (i) G' is the subgraph according to Def. 2 for C. (ii) There exists a vertex v_c in G' labeled with $c \in C$ and an edge e that is an instance of p and c induced by [rule] such that removal of e from G' makes v_c backward unreachable from $v_\square$. In the graphs below e and v_c are marked blue, thick, dashed. The vertex labels use `TRP++` syntax.

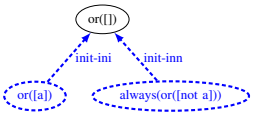
Note that the search for such triples can be supported by a corresponding modification to the temporal resolution solver. For the more complex cases below candidates were obtained in that way and then optimized by hand.

[2]Note that this proof assumes that there are no edges between instances of the premise of [aug2], the premises of [BFS-loop-it-init-c], and premise 2 of [BFS-loop-conclusion2] and their conclusions induced by these production rules as indicated in Tab. I and Def. 1. I.e., different minimal sets of premises to include in the resolution graph may exist.

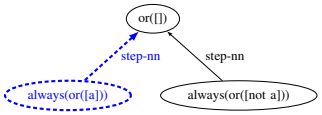
[init-ii] $\{(a), (\neg a)\}$



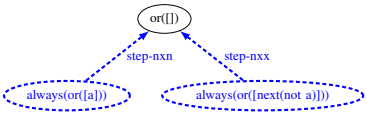
[init-in], premises 1, 2 $\{(a), (\mathbf{G}(\neg a))\}$



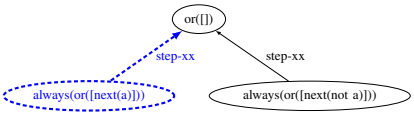
[step-nn] $\{(\mathbf{G}(a)), (\mathbf{G}(\neg a))\}$



[step-nx], premises 1, 2 $\{(\mathbf{G}(a)), (\mathbf{G}(\mathbf{X}(\neg a)))\}$



[step-xx] $\{(\mathbf{G}(\mathbf{X}(a))), (\mathbf{G}(\mathbf{X}(\neg a)))\}$



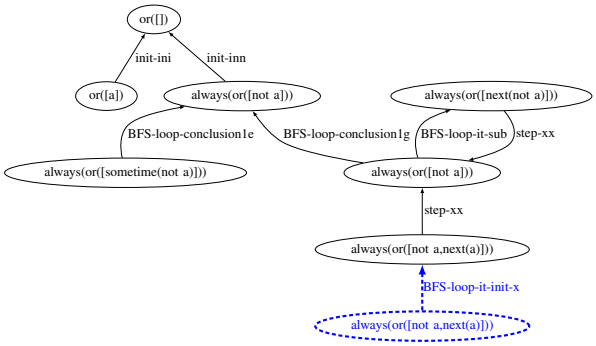
aug1

$$\{(a),\ (\mathbf{G}((\neg b) \vee (\mathbf{X}(\neg c)))),\ (\mathbf{G}((\neg c) \vee (\mathbf{X}(c)))),\ (\mathbf{G}((\neg a) \vee (\mathbf{F}(\neg c)))),\ (\mathbf{G}((c) \vee (\mathbf{F}(b)))),\ (\mathbf{G}(\mathbf{F}(c)))\}$$

See Fig. 6.

BFS-loop-it-init-x

$$\{(a),\ (\mathbf{G}((\neg a) \vee (\mathbf{X}(a)))),\ (\mathbf{G}(\mathbf{F}(\neg a)))\}$$



BFS-loop-it-init-n

$$\{(a),\ (\mathbf{G}((\neg b) \vee a)),\ (\mathbf{G}((\neg b) \vee (\neg a))),\ (\mathbf{G}((\neg a) \vee (\mathbf{F}(b))))\}$$

See Fig. 7.

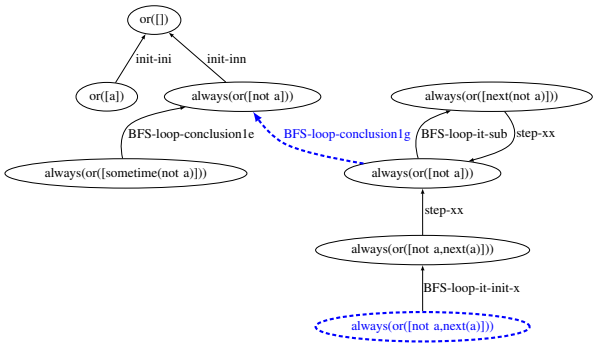
BFS-loop-it-sub

$$\{(a),\ (\mathbf{G}((\neg a) \vee (\mathbf{X}(b)))),\ (\mathbf{G}((\neg b) \vee (\mathbf{X}(a)))),\ (\mathbf{G}(\mathbf{F}(c))),\ (\mathbf{G}((\neg c) \vee (\neg a))),\ (\mathbf{G}((\neg a) \vee (\mathbf{X}(\neg c))))\}$$

See Fig. 8.

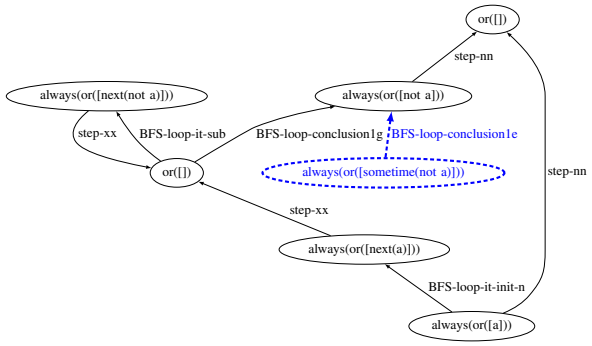
BFS-loop-conclusion1, premise 1

$$\{(a),\ (\mathbf{G}((\neg a) \vee (\mathbf{X}(a)))),\ (\mathbf{G}(\mathbf{F}(\neg a)))\}$$



BFS-loop-conclusion1, premise 2

$$\{(\mathbf{G}(a)),\ (\mathbf{G}(\mathbf{F}(\neg a)))\}$$



BFS-loop-conclusion2, premise 1

$$\{(a \vee b),\ (\mathbf{G}((\neg a) \vee (\mathbf{F}(f)))),\ (\mathbf{G}((\neg b) \vee (\mathbf{F}(f)))),\ (\mathbf{G}((\neg a) \vee (\mathbf{F}(\neg f)))),\ (\mathbf{G}((\neg b) \vee (\mathbf{F}(\neg f)))),\ (\mathbf{G}(((\neg a) \vee f) \vee (\mathbf{X}(c)))),\ (\mathbf{G}(((\neg b) \vee f) \vee (\mathbf{X}(d)))),\ (\mathbf{G}((\neg c) \vee (\mathbf{X}(c)))),\ (\mathbf{G}((\neg d) \vee (\mathbf{X}(d)))),\ (\mathbf{G}((\neg c) \vee (\neg f))),\ (\mathbf{G}((\neg d) \vee (\neg f))),\ ((\neg f) \vee g),\ (\mathbf{G}((\neg g) \vee (\mathbf{X}(g)))),\ (\mathbf{G}((\neg g) \vee f))\}$$

See Fig. 9.

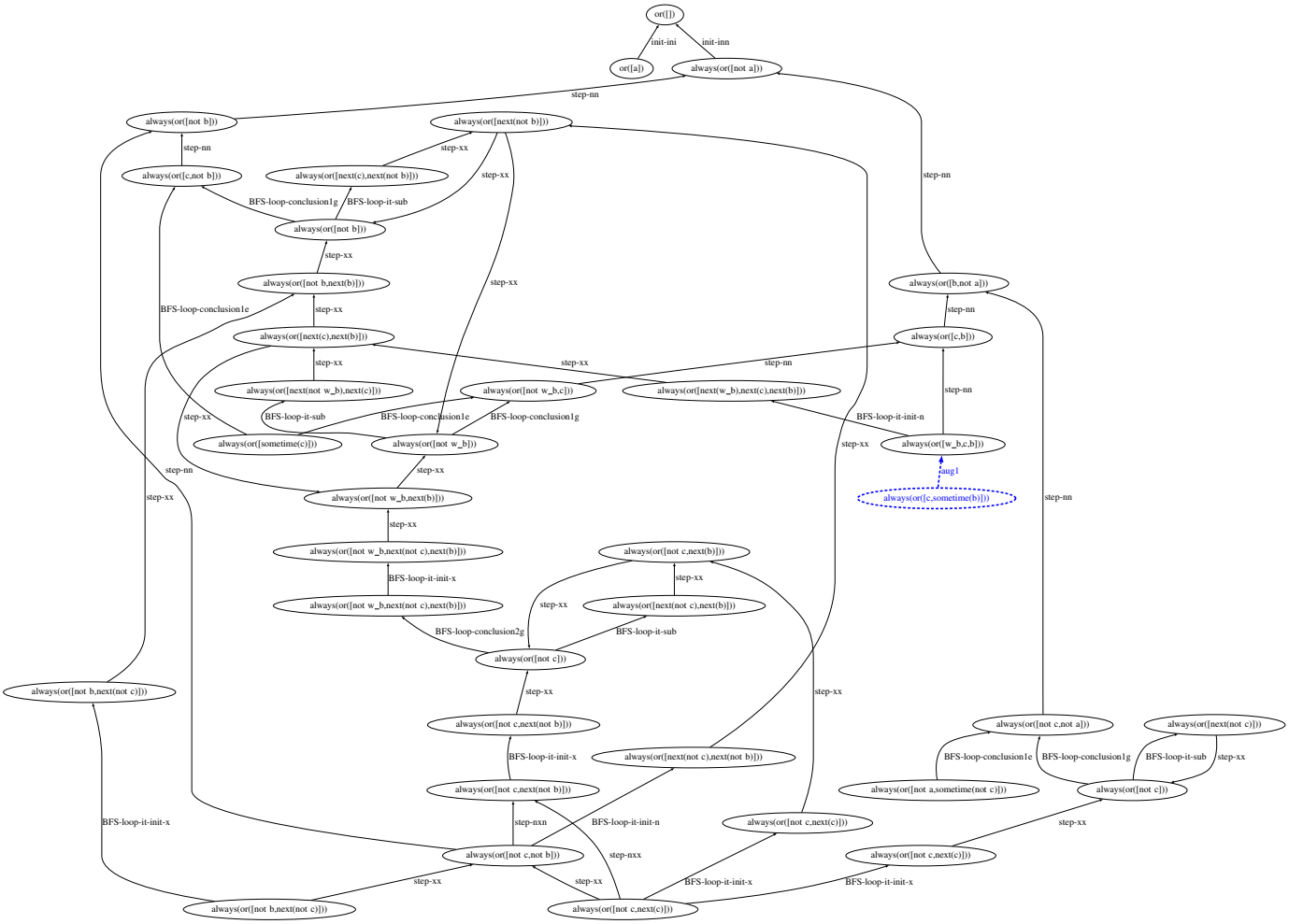


Fig. 6. Subgraph G' of resolution graph for the case of aug1 in the proof of Remark 1.

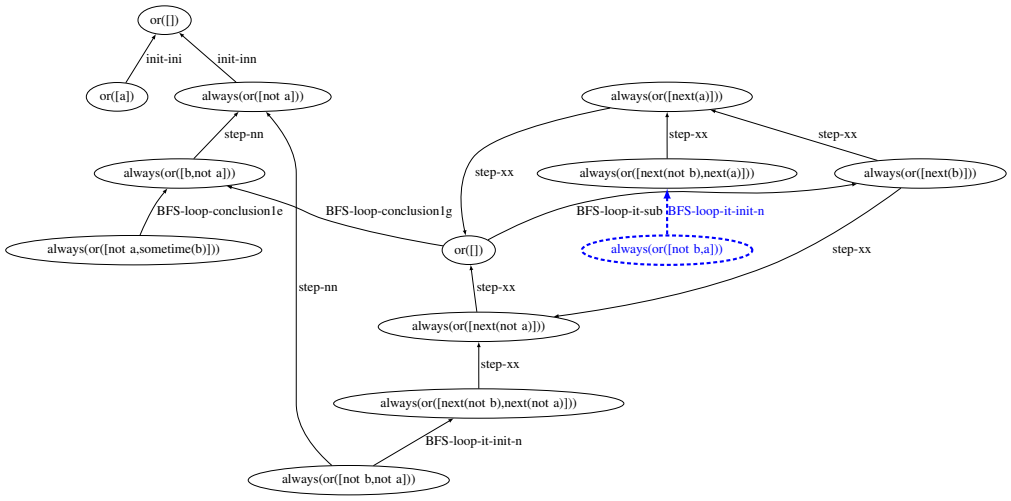


Fig. 7. Subgraph G' of resolution graph for the case of BFS-loop-it-init-n in the proof of Remark 1.

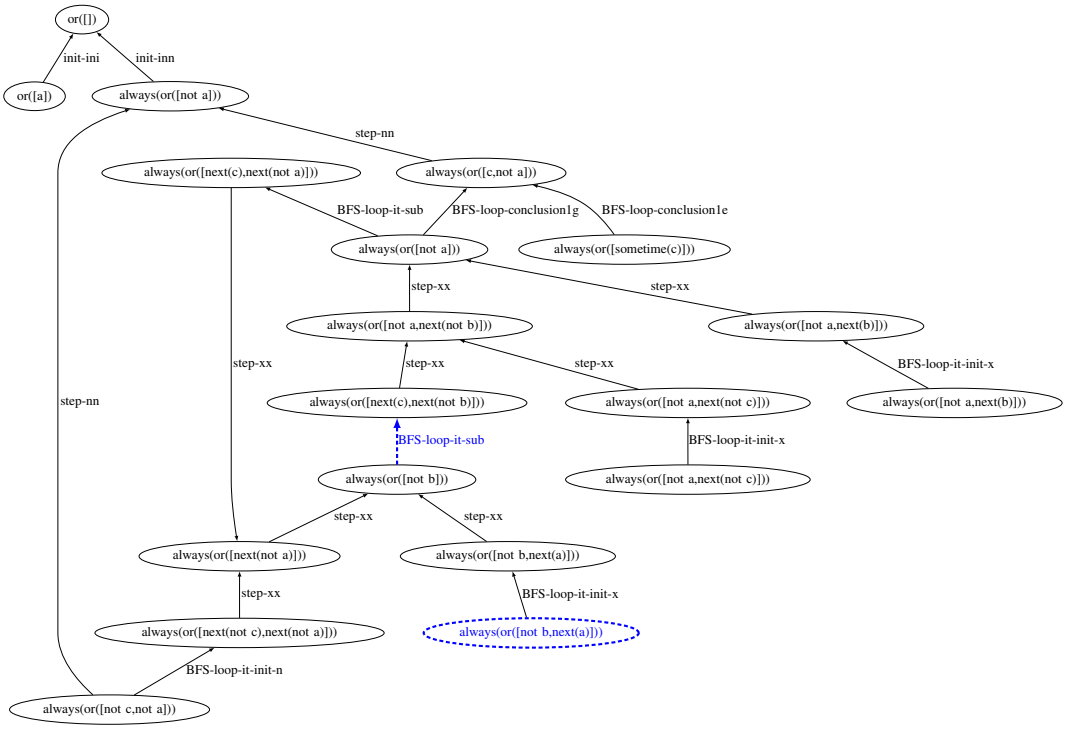


Fig. 8. Subgraph G' of resolution graph for the case of BFS-loop-it-sub in the proof of Remark 1.

This concludes the proof. ■

APPENDIX B
PROOFS: V FROM LTL TO SNF AND BACK

Theorem 2 (Unsatisfiability of UC in LTL)**.** *Let ϕ be an unsatisfiable LTL formula, and let ϕ^{uc} be the UC of ϕ in LTL. Then ϕ^{uc} is unsatisfiable.*

Proof: Let $SNF(\phi)$ be the SNF of ϕ, and let C^{uc} be the UC of $SNF(\phi)$ in SNF.

First, consider the trivial case that ϕ is 0. Here, Def. 4 results in the UC of ϕ in LTL being $\phi^{uc} \equiv 0$ as desired.

Now assume that ϕ is not 0, i.e., the size of the syntax tree of ϕ is greater than 1. Let $SNF(\phi^{uc})$ be the SNF of ϕ^{uc}. In order to prove that ϕ^{uc} is unsatisfiable we show that the clauses of C^{uc} (which is unsatisfiable) are a subset of the SNF of ϕ^{uc}: $C^{uc} \subseteq SNF(\phi^{uc})$.

By comparing the clauses of $SNF(\phi)$ with those of $SNF(\phi^{uc})$ we can partition the clauses of $SNF(\phi)$ into 3 sets:[3] (i) Some clauses are present in both $SNF(\phi)$ and $SNF(\phi^{uc})$: $C'_1 \equiv SNF(\phi) \cap SNF(\phi^{uc})$. (ii) Some clauses are present in $SNF(\phi)$ and are present in $SNF(\phi^{uc})$ with one or more occurrences of some propositions $x, x', \ldots$ that are marked blue boxed in Tab. II replaced with 1 or 0. Call that set C'_2. (iii) Some clauses are present in $SNF(\phi)$ but not in $SNF(\phi^{uc})$: $C'_3 \equiv SNF(\phi) \setminus (SNF(\phi^{uc}) \cup C'_2)$.

[3]We disregard the issue of the indices of the variables $x, x', \ldots$.

By Def. 2 C^{uc} is a subset of $SNF(\phi)$: $C^{uc} \subseteq SNF(\phi)$. By Def. 4 C^{uc} contains no member of C'_2; otherwise, there could not be one or more occurrences of some propositions $x, x', \ldots$ that are marked blue boxed in Tab. II replaced with 1 or 0 in the clauses of C'_2: $C^{uc} \cap C'_2 = \emptyset$. Now we argue that C^{uc} also contains no member of C'_3. First, let $c \in C'_3$ be an initial or a global clause. c cannot be a member of C^{uc} as, in order to be part of a proof that derives the empty clause, all literals of c need to be "resolved away". However, this is not possible for c as for the literal $(\neg)x_\psi$ on the left side of the implication in Tab. II there is no clause with an opposite literal in C^{uc}. This follows by induction on the nesting depth of the subformula ψ to which $(\neg)x_\psi$ belongs from the occurrence of the superformula of ψ that has been replaced with 1 or 0 in ϕ^{uc}. Now let $c \in C'_3$ be an eventuality clause. By Def. 1, 2 for such c to be part of C^{uc} there would have to be a clause c' in the resolution graph G according to Def. 1 that was generated by production rules aug1 or BFS-loop-conclusion1 and that is backward reachable in G from the vertex labeled with the empty clause $\square$ in the main partition M, $v_\square$. Again, for the latter to happen, all literals of c' would have to be "resolved away", which is impossible by a similar inductive argument as before. Hence, we have shown that all clauses in C^{uc} come from C'_1, which is a subset of $SNF(\phi^{uc})$. This concludes the proof. ■

APPENDIX C
PROOFS: VI MINIMAL UCS

Remark 4 (Minimal UC in SNF No Guarantee for Minimal UC in LTL)**.** *Let ϕ be an unsatisfiable LTL formula, C its*

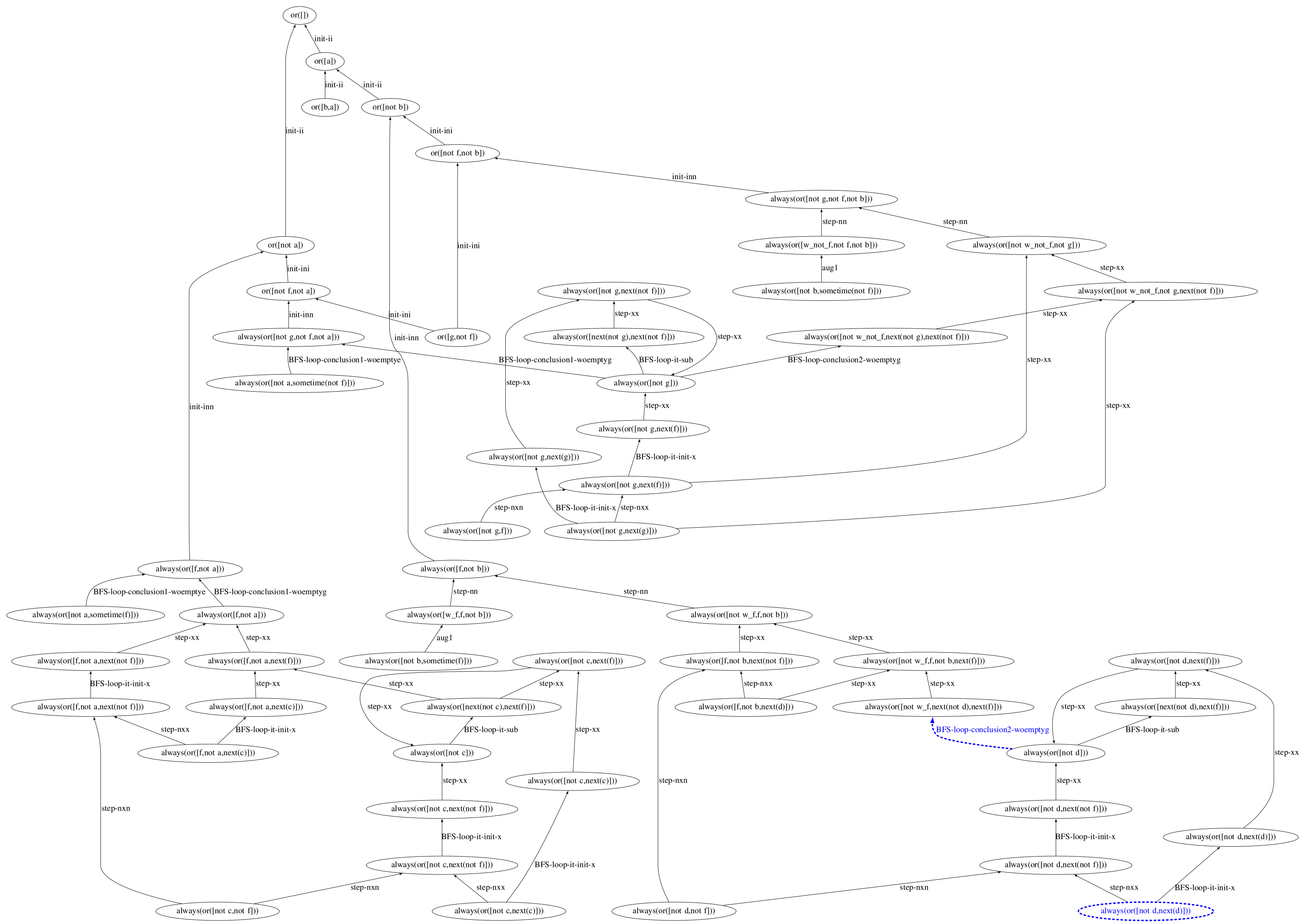


Fig. 9. Subgraph G' of resolution graph for the case of BFS-loop-conclusion2 in the proof of Remark 1.

translation to SNF, C^{uc} a minimal UC of C in SNF, and ϕ^{uc} the UC of ϕ in LTL obtained by mapping C^{uc} back to LTL via Def. 4. Then ϕ^{uc} is not necessarily minimal.

Proof: Let $\phi \equiv (\neg p) \wedge ((\mathbf{G}\neg q) \wedge (p\mathbf{U}q))$. Then

$$C \equiv \begin{array}{l}\{x_\phi,\\ (\mathbf{G}(x_\phi \rightarrow x_{\neg p})),\\ (\mathbf{G}(x_{\neg p} \rightarrow \neg p)),\\ (\mathbf{G}(x_\phi \rightarrow x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)})),\\ (\mathbf{G}(x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)} \rightarrow x_{\mathbf{G}\neg q})),\\ (\mathbf{G}(x_{\mathbf{G}\neg q} \rightarrow \mathbf{X}x_{\mathbf{G}\neg q})),\\ (\mathbf{G}(x_{\mathbf{G}\neg q} \rightarrow x_{\neg q})),\\ (\mathbf{G}(x_{\neg q} \rightarrow \neg q)),\\ (\mathbf{G}(x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)} \rightarrow x_{p\mathbf{U}q})),\\ (\mathbf{G}(x_{p\mathbf{U}q} \rightarrow (q \vee p))),\\ (\mathbf{G}(x_{p\mathbf{U}q} \rightarrow (q \vee \mathbf{X}x_{x_{p\mathbf{U}q}}))),\\ (\mathbf{G}(x_{p\mathbf{U}q} \rightarrow \mathbf{F}q))\}.\end{array}$$

is its SNF according to Def. 3. A minimal UC of C in SNF is

$$C^{uc} \equiv \begin{array}{l}\{x_\phi,\\ (\mathbf{G}(x_\phi \rightarrow x_{\neg p})),\\ (\mathbf{G}(x_{\neg p} \rightarrow \neg p)),\\ (\mathbf{G}(x_\phi \rightarrow x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)})),\\ (\mathbf{G}(x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)} \rightarrow x_{\mathbf{G}\neg q})),\\ (\mathbf{G}(x_{\mathbf{G}\neg q} \rightarrow x_{\neg q})),\\ (\mathbf{G}(x_{\neg q} \rightarrow \neg q)),\\ (\mathbf{G}(x_{(\mathbf{G}\neg q)\wedge(p\mathbf{U}q)} \rightarrow x_{p\mathbf{U}q})),\\ (\mathbf{G}(x_{p\mathbf{U}q} \rightarrow (q \vee p)))\}.\end{array}$$

Mapping C^{uc} back to a UC in LTL via Def. 4 yields ϕ. ϕ is not a minimal UC, as the first conjunct $\neg p$ can be replaced with 1 while retaining unsatisfiability.

Note that given ϕ our implementation actually produces ϕ as a UC in LTL. This is due to the fact that the UC in SNF, C^{uc}, is found during the first execution of saturation in line 2 of the algorithm in Fig. 2, while the contradiction between $\mathbf{G}\neg q$ and the eventuality part of $p\mathbf{U}q$ requires loop search, which is only performed at a later stage.

Note also that the result just proved depends on the notion of UC for LTL: the proof above obviously does not hold if the notion of UC allows to not only replace $\mathbf{G}\neg q$ with 1 but also, alternatively, with $\neg q$ and $p\mathbf{U}q$ not only with 1 but also, alternatively, with $p \vee q$. ■

Appendix D
Relation to Mutual Vacuity

In [GC04] Gurfinkel and Chechik introduce the notion of mutual vacuity. It is easy to see that the problems of finding a UC of an unsatisfiable formula ϕ in LTL and finding a set of subformula occurrences $\mathcal{O}$ of an LTL specification ϕ such that ϕ is mutually vacuously 1 in $\mathcal{O}$ in a system ζ can conceptually be reduced to each other. For some more discussion on the relation between UCs and vacuity see also [Sch12].

Definition 6 (Mutual Vacuity)**.** *Let ζ be a system description. Let ϕ be a specification in LTL such that ϕ is 1 in ζ. Let $\mathcal{O}$ be a set of disjoint subformula occurrences in ϕ. Then ϕ is* mutually vacuously 1 *in $\mathcal{O}$ in ζ iff the modification ϕ' of ϕ that replaces those members of $\mathcal{O}$ that have positive (resp. negative) polarity in ϕ with 0 (resp. 1) is 1 in ζ.*

Remark 6 (LTL Model Checking as LTL Satisfiability)**.** *Let ζ be a system description in LTL. Let ϕ be a specification in LTL. Then ϕ is 1 in ζ iff $\zeta \wedge \neg\phi$ is unsatisfiable.*

Note that frequently system descriptions can be translated into LTL.

Remark 7 (Reducibility between Mutual Vacuity and UCs in LTL)**.** *The problems of finding a UC of an unsatisfiable formula ϕ in LTL and finding a set of subformula occurrences $\mathcal{O}$ of an LTL specification ϕ that is 1 in an LTL system description ζ such that ϕ is mutually vacuously 1 in $\mathcal{O}$ in ζ can be reduced to each other.*

Proof: The proof is essentially by the respective definitions.

Assume ϕ is mutually vacuously 1 in $\mathcal{O}$ in ζ. Let ϕ' be ϕ with positive (resp. negative) polarity members of $\mathcal{O}$ replaced with 0 (resp. 1). Then (i) $\mu \equiv \zeta \wedge \neg\phi$ is unsatisfiable. (ii) $\mu' \equiv \zeta \wedge \neg\phi'$ is unsatisfiable. μ' is a UC of μ in LTL.

Assume ϕ is an unsatisfiable LTL formula with UC ϕ^{uc}. Then there exists a non-empty set of subformula occurrences $\mathcal{O}'$ in ϕ such that ϕ^{uc} is obtained from ϕ by replacing positive (resp. negative) polarity members of $\mathcal{O}'$ with 1 (resp. 0). Now obviously (i) $1 \wedge \neg\neg\phi$ is unsatisfiable and (ii) $1 \wedge \neg\neg\phi^{uc}$ is unsatisfiable. I.e., $\neg\phi$ is mutually vacuously 1 in $\mathcal{O}'$ in the unconstrained system 1. ■

A limited number of tools have been made available that can determine vacuity. `Aardvark` [PWK09] computes — depending on configuration — maximal or maximum mutual vacuity for LTL using `VIS` [vis96] as a backend. `Aardvark` uses binary search and counterexamples to reduce the search space in the lattice of candidate strengthened specifications; it does not use proofs for passing specifications to obtain an initial candidate strengthened specification. Hence, our method is complementary. We performed a small set of trials with `Aardvark` on some of our benchmarks, mostly the smaller ones from each family. Within this, admittedly limited, set of trials we ran into problems with usability (often getting assertion violations or segmentation faults rather than error messages pointing to potential problems in our input) as well as scalability.[4] We therefore opted not to perform a comparison on the full set of benchmarks. `VaqTree` [SDG06] implements the method of [Sim+10] that computes k-step vacuity for LTL, i.e., whether an occurrence of an atomic proposition is vacuous

[4]Note that in vacuity checking it is typically assumed that the system description is more complex than the specification, while in UC extraction all complexity is in the formula at hand. When (as we did in our trials) using the reduction from LTL UC extraction to vacuity checking from Rem. 7 the resulting vacuity checking instance consists of a trivial system description and a complex specification. In practice, when the vacuity checking procedure is tuned to take advantage of the small specification/complex system description scenario, then complex specifications may lead to problems; it seems that the scalability problems we observed with `Aardvark` are caused to some extent by the translation from the LTL specification to an explicit Büchi automaton performed in `VIS`.

when bounded model checking runs only up to some bound k are considered; the problem of removing the bound k is left open. The `NuSMV Model Advisor` [nma] computes the set of all occurrences of atomic propositions that are vacuous (but not necessarily mutually vacuous) for LTL. `VaqUoT` [GG06], a simplified implementation of [GC04], computes the set of all occurrences of atomic propositions that are vacuous (but not necessarily mutually vacuous) for CTL. χ`Chek` [Eas+03] is a multi-valued model checker for CTL. A proof-based formulation of vacuity is suggested by Namjoshi [Nam04]; no implementation or experiments are reported.

Appendix E
A Complete Example

In this section we present a complete example: we start with an LTL formula, translate this into SNF according to Def. 3, perform TR on the SNF as described in Sec. III, extract a UC in SNF via Def. 1, 2, and finally obtain a (proper) UC of the original LTL formula by Def. 4.

The formula ϕ we would like to obtain a UC of is (15). Clearly, ϕ is unsatisfiable. Moreover, q can be replaced with 1 without making ϕ satisfiable. q is the only subformula that is not required for unsatisfiability.

$$\phi \equiv \mathbf{G}(p \wedge q) \wedge \mathbf{F}\neg p \tag{15}$$

When translating ϕ into a set of SNF clauses C our implementation treats top level conjuncts as separate formulas. We therefore separately translate $\mathbf{G}(p \wedge q)$ and $\mathbf{F}\neg p$ according to Def. 3. $\mathbf{G}(p \wedge q)$ is translated into 5 clauses x_1, $\mathbf{G}(\neg x_1 \vee \mathbf{X}x_1)$, $\mathbf{G}(\neg x_1 \vee x_2)$, $\mathbf{G}(\neg x_2 \vee p)$, and $\mathbf{G}(\neg x_2 \vee q)$. x_1 represents $\mathbf{G}(p \wedge q)$ in the sense that if in a satisfying assignment of C x_1 is 1, then $\mathbf{G}(p \wedge q)$ is 1. Similarly, x_2 represents $p \wedge q$. The translation of $\mathbf{F}\neg p$ results in 3 clauses x_3, $\mathbf{G}(\neg x_3 \vee \mathbf{F}x_4)$, and $\mathbf{G}(\neg p \vee \neg x_4)$. x_3 stands for $\mathbf{F}\neg p$ and x_4 for $\neg p$. The SNF of ϕ, C, is shown in (16).

$$\begin{aligned} C \equiv \{&x_1, \mathbf{G}(\neg x_1 \vee \mathbf{X}x_1), \mathbf{G}(\neg x_1 \vee x_2),\\ &\mathbf{G}(\neg x_2 \vee p), \mathbf{G}(\neg x_2 \vee q), x_3,\\ &\mathbf{G}(\neg x_3 \vee \mathbf{F}x_4), \mathbf{G}(\neg p \vee \neg x_4)\} \end{aligned} \tag{16}$$

In Fig. 10 we show an execution of the algorithm in Fig. 2 on C. In Fig. 10 TR generally proceeds from bottom to top. At the bottom are the clauses in C. The leftmost clause in the top row is the empty clause $\square$, indicating unsatisfiability. Clauses are connected with directed edges from premises to conclusions according to columns 9, 10 in Tab. I. Edges are labeled with production rules, where "BFS-loop" is abbreviated to "loop", "init" to "i", and "conclusion" to "conc". Saturation in line 2 of the algorithm in Fig. 2 produces no new clauses.[5] The 2 clauses in row 2 are generated by augmentation (line 3). The following saturation (line 4) produces no new clauses. The dark green shaded rectangle is the loop partition for the first loop search iteration. Row 3 contains the clauses obtained by initialization of the BFS loop search iteration (line 11). Row 4 then contains the clauses generated from those in row 3 by saturation restricted to step-xx (line 12). The subsumption test fails in this iteration, as $\neg x_1$ (from $\mathbf{G}(\neg x_1)$) does not subsume $\square$ (from $\mathbf{G}(\mathbf{X}x_4)$) (lines 13–15). The light green shaded rectangle is the loop partition for the second loop search iteration. Row 5 contains the clauses obtained by initialization and row 6 those obtained from them by restricted saturation. This time the subsumption test succeeds, and the loop search conclusions are shown in row 7 (line 18). Finally, while row 8 contains a "blind alley", row 9 has the derivation of the empty clause $\square$ via saturation (line 19).

The thick, dotted, blue clauses and edges show the part of the resolution graph that is backward reachable from $\square$. Clause $\mathbf{G}(\neg x_2 \vee q)$ is the only clause of C that is not backward reachable from $\square$. Hence, the UC of ϕ in SNF according to Def. 1, 2 is $C^{uc} = C \setminus \{\mathbf{G}(\neg x_2 \vee q)\}$ as shown in (17).

$$\begin{aligned} C^{uc} \equiv \{&x_1, \mathbf{G}(\neg x_1 \vee \mathbf{X}x_1), \mathbf{G}(\neg x_1 \vee x_2), \mathbf{G}(\neg x_2 \vee p),\\ &x_3, \mathbf{G}(\neg x_3 \vee \mathbf{F}x_4), \mathbf{G}(\neg p \vee \neg x_4)\} \end{aligned} \tag{17}$$

By careful inspection we see that q is the only subformula whose proposition according to Tab. II (which is q itself) does not occur in any clause of C^{uc} in a position that is marked blue boxed in Tab. II. Hence, q is the only subformula to be replaced by 1 or 0 in ϕ, yielding the UC of ϕ in LTL, ϕ^{uc}, in (18).

$$\phi^{uc} \equiv \mathbf{G}(p \wedge 1) \wedge \mathbf{F}\neg p \tag{18}$$

Appendix F
Additional Plots

Figures 11 and 12 show the overhead that is incurred and the size reduction that is obtained by extracting (non-minimal) UCs compared to not extracting UCs split by category.

Figures 13 and 14 show the overhead that is incurred and the size reduction that is obtained by extracting minimal UCs compared to extracting (non-minimal) UCs split by category.

Figures 15 and 16 show the benefit of optimizations split by category.

[5] While it may seem that some clauses are not considered for saturation, this is due to either subsumption of one clause by another (e.g., $\mathbf{G}(\neg wx_4 \vee \mathbf{X}\neg x_1 \vee \mathbf{X}x_4)$ obtained from $\mathbf{G}(\neg wx_4 \vee \mathbf{X}x_4 \vee \mathbf{X}wx_4)$ and $\mathbf{G}(\neg x_1 \vee \neg wx_4)$ is subsumed by $\mathbf{G}(\neg wx_4 \vee \mathbf{X}\neg x_1)$) or the fact that `TRP++` uses *ordered* resolution (e.g., x_1 with $\mathbf{G}(\neg x_1 \vee x_2)$ — the order here is $x_1 < x_2 < p < q < x_3 < x_4$; [HK03,BG01]). Both are issues of completeness of TR and, therefore, not discussed in this paper.

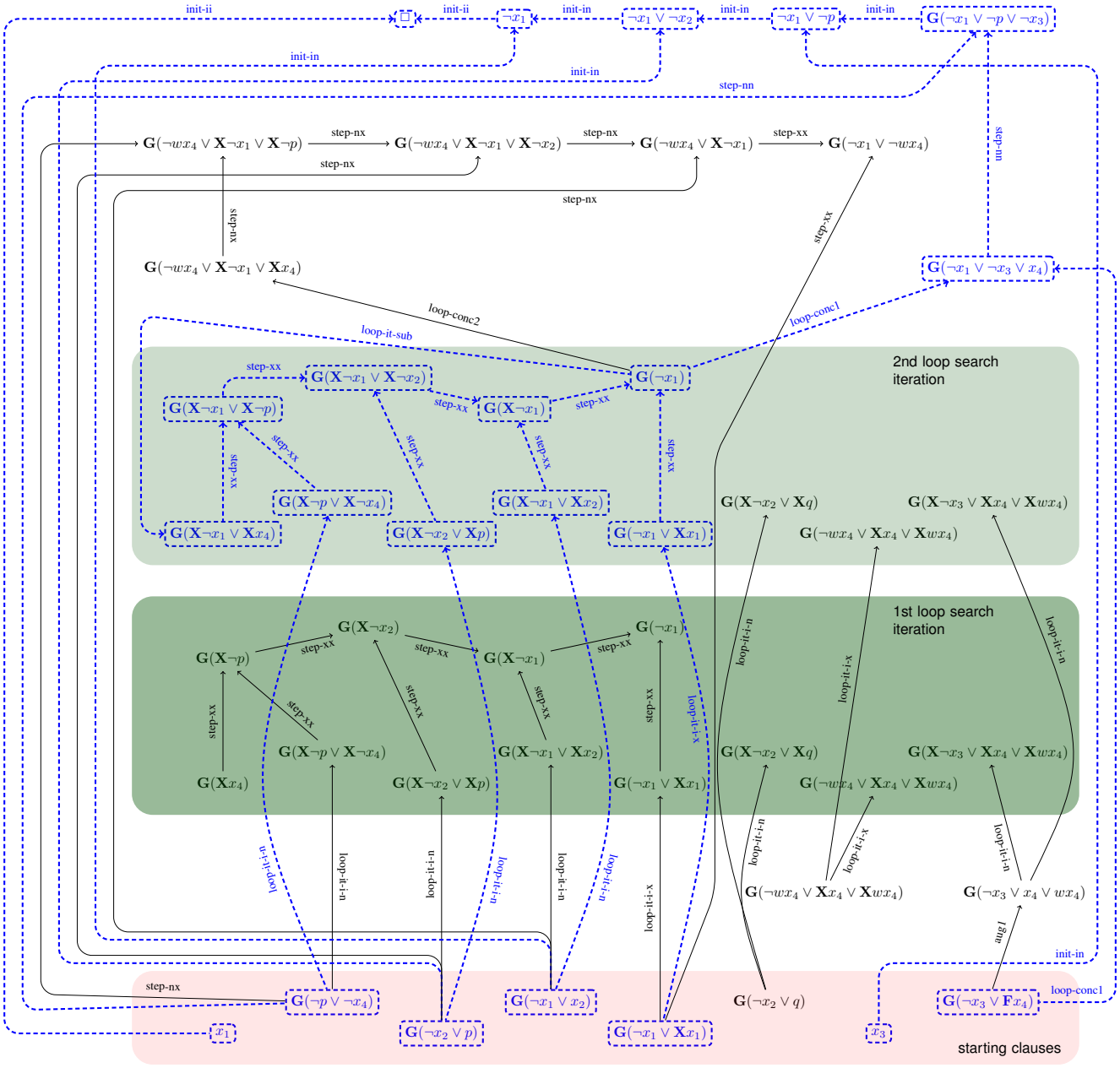


Fig. 10. Example of an execution of the TR algorithm with corresponding resolution graph and UC extraction in SNF.

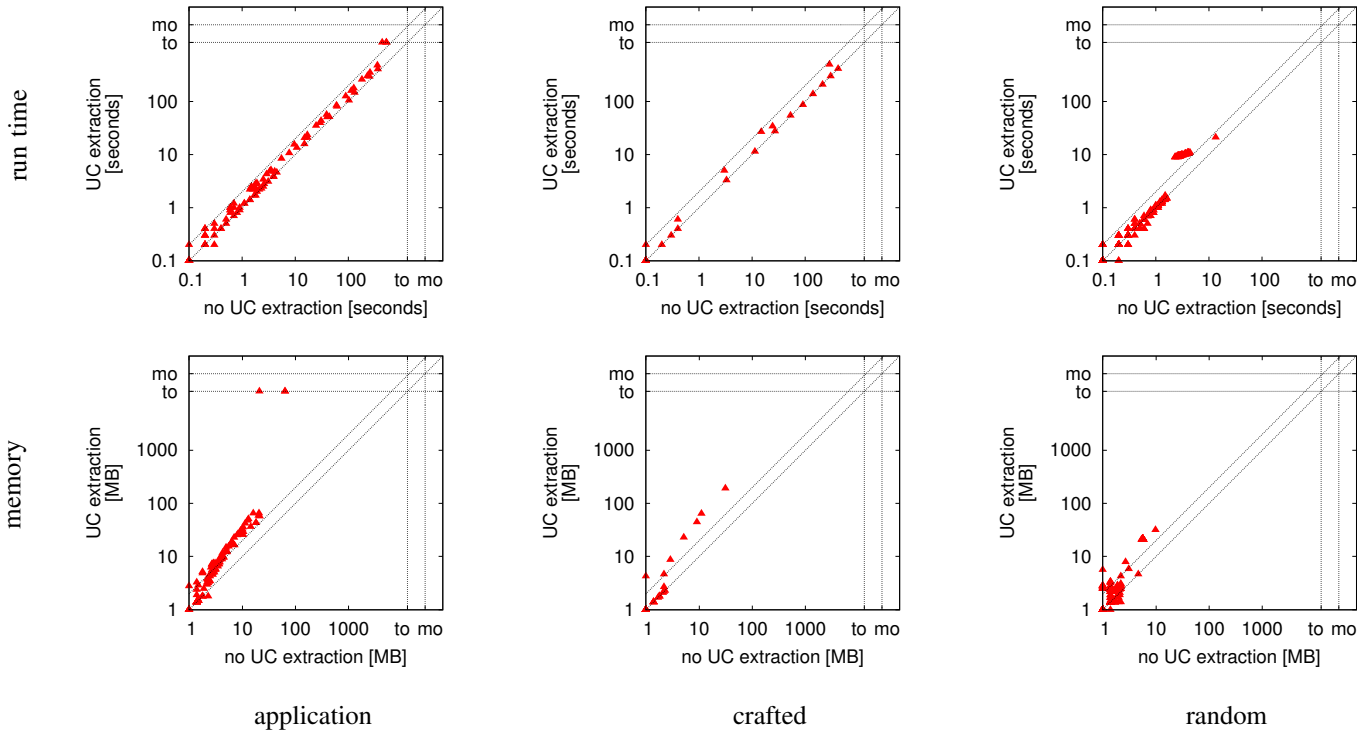


Fig. 11. Overhead incurred by (non-minimal) UC extraction compared to not extracting UCs in terms of run time (in seconds) and memory (in MB) separated by categories **application**, **crafted**, and **random**. In each graph extraction of UCs is on the y-axis and no UC extraction on the x-axis. The off-center diagonal shows where $y = 2x$.

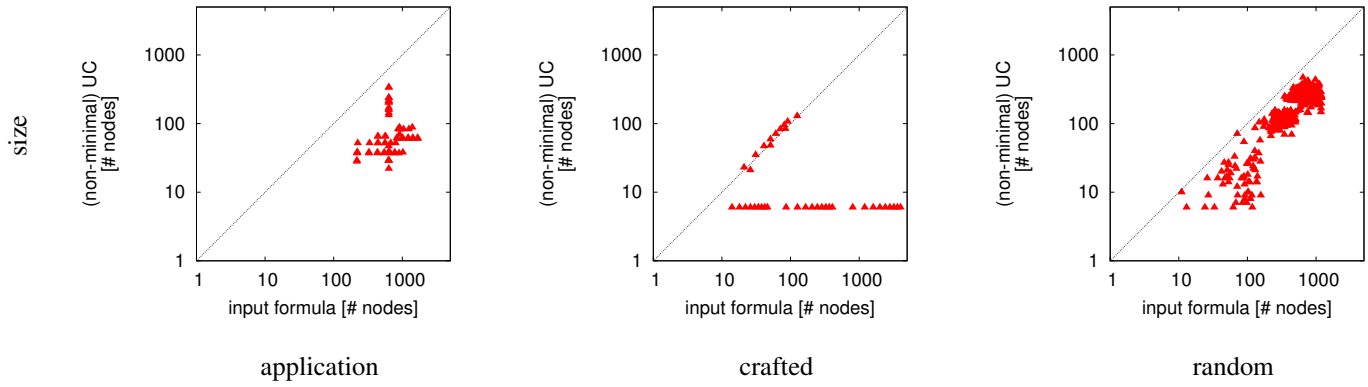


Fig. 12. Size reduction obtained by (non-minimal) UC extraction compared to not extracting UCs separated by categories **application**, **crafted**, and **random**. The y-axes show the sizes of the (non-minimal) UCs, the x-axes show the sizes of the input formulas. Size is measured as the number of nodes in the syntax trees.

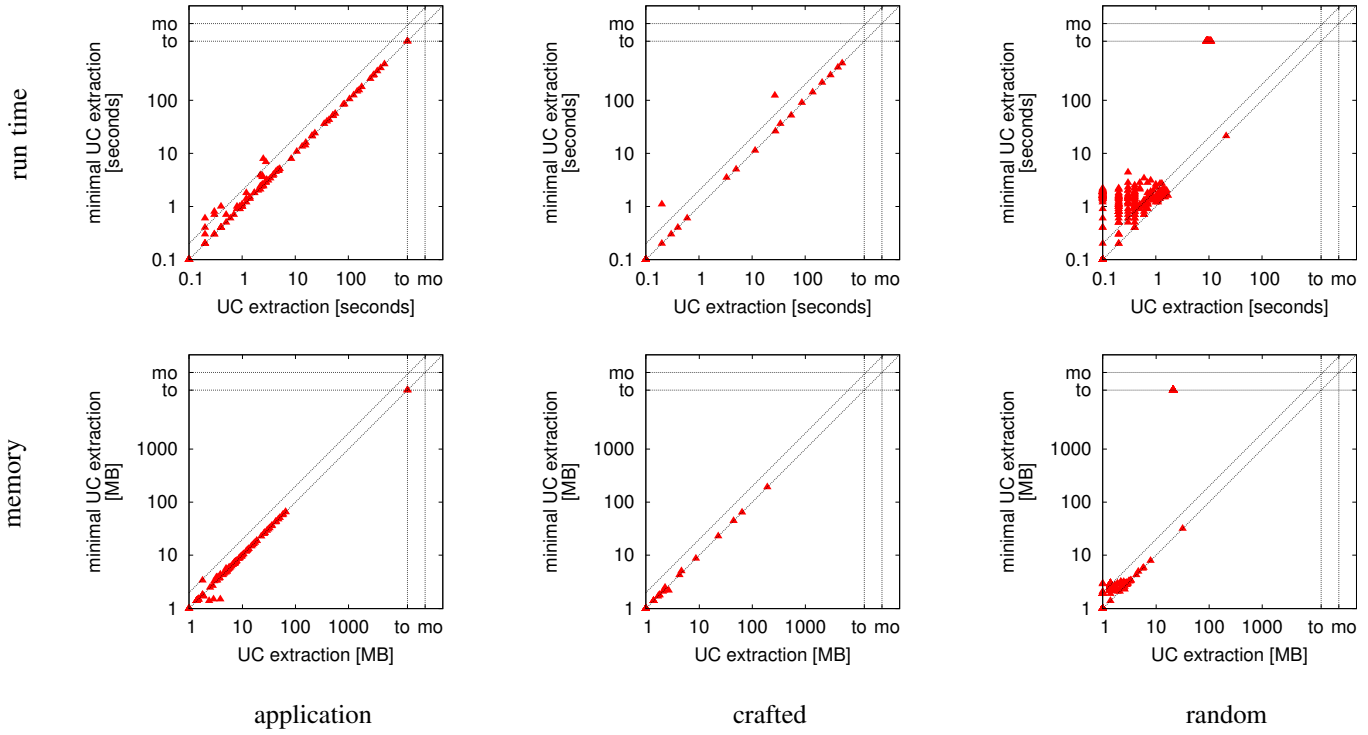


Fig. 13. Overhead incurred by minimal UC extraction compared to (non-minimal) UC extraction in terms of run time (in seconds) and memory (in MB) separated by categories **application**, **crafted**, and **random**. In each graph extraction of minimal UCs is on the y-axis and (non-minimal) UC extraction on the x-axis. The off-center diagonal shows where $y = 2x$.

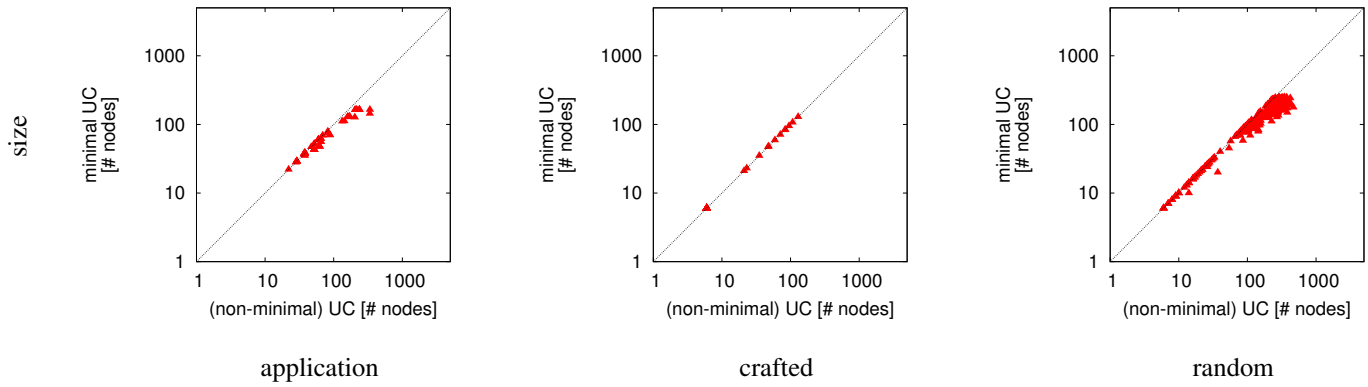


Fig. 14. Size reduction obtained by minimal UC extraction compared to (non-minimal) UC extraction separated by categories **application**, **crafted**, and **random**. The y-axes show the sizes of the minimal UCs, the x-axes show the sizes of the (non-minimal) UCs. Size is measured as the number of nodes in the syntax trees.

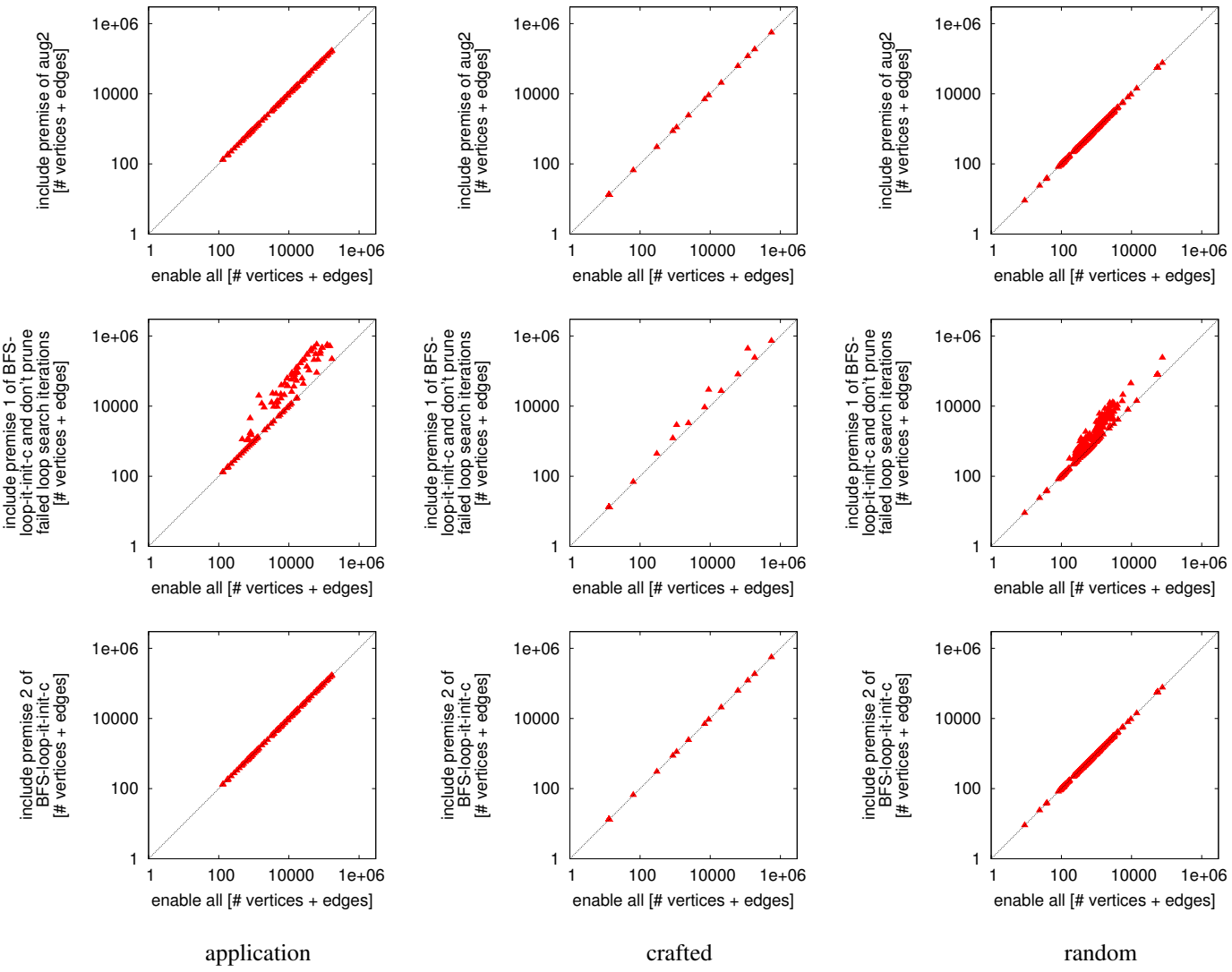


Fig. 15. Benefit of optimizations as reduction in peak size of resolution graph (number of vertices + number of edges) separated by categories **application**, **crafted**, and **random**. The x-axis shows all optimizations enabled. The y-axis of rows 1–3 shows one optimization disabled: (row 1) include premise of aug2, (row 2) include premise 1 of BFS-loop-it-init-c and disable immediate pruning of failed loop search iterations, and (row 3) include premise 2 of BFS-loop-it-init-c.

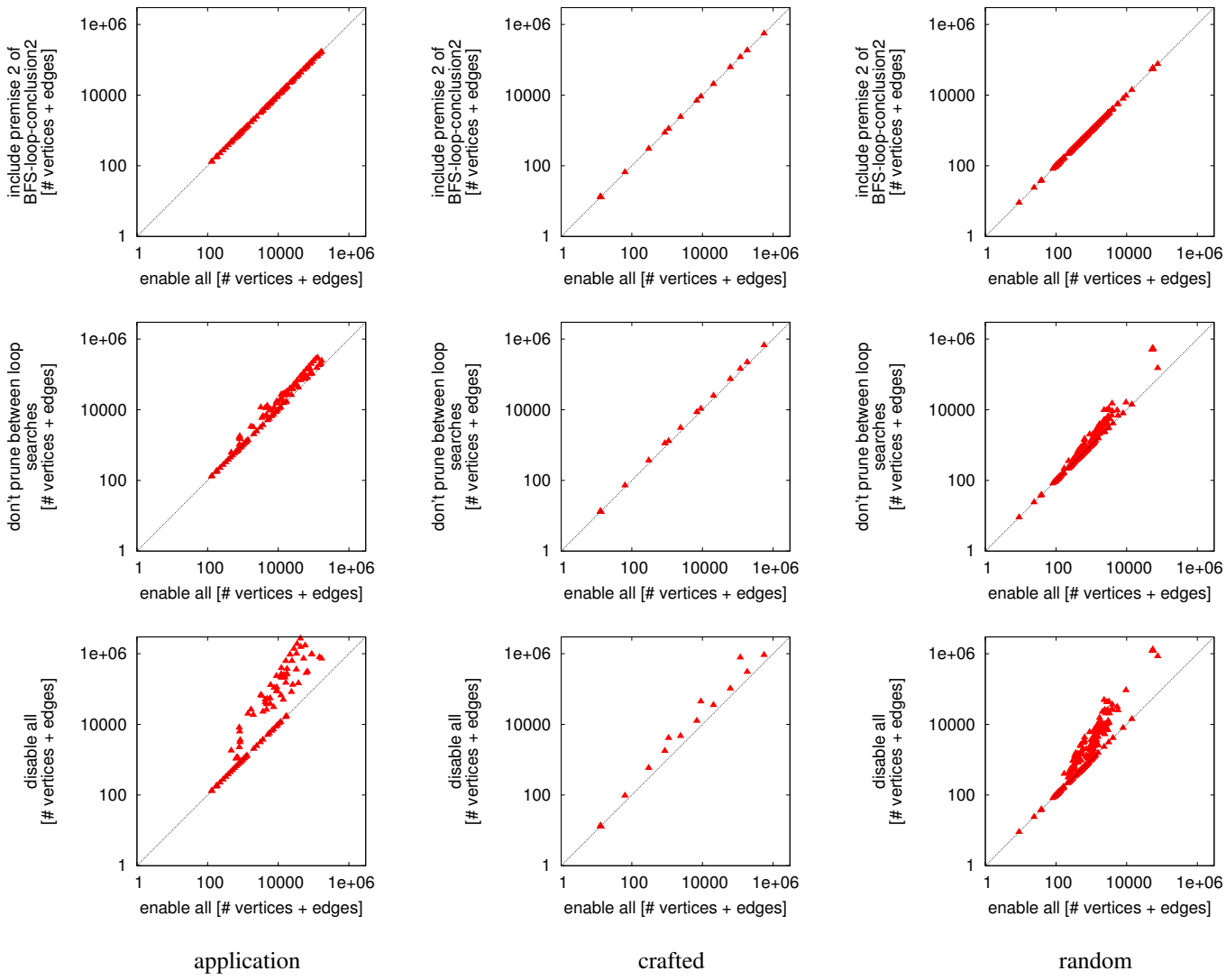


Fig. 16. Benefit of optimizations as reduction in peak size of resolution graph (number of vertices + number of edges) separated by categories **application**, **crafted**, and **random**. The x-axis shows all optimizations enabled. The y-axis of rows 1 and 2 shows one optimization disabled: (row 1) include premise 2 of BFS-loop-conclusion2 and (row 2) disable pruning of the resolution graph between loop searches. The y-axis of row 3 shows all optimizations disabled.